Thermal and Power Sensing and Management for Mobile System-On-a-Chip

by
Sofiane Chetoui

M.Sc., Brown University, Providence, RI, 2020
State Engineering Degree, M.Sc., Ecole Nationale Polytechnique, Algeria, 2017

A dissertation submitted in partial fulfillment of the
requirements for the degree of Doctor of Philosophy
in School of Engineering at Brown University

PROVIDENCE, RHODE ISLAND

April 2022

This dissertation by Sofiane Chetoui is accepted in its present form
by School of Engineering as satisfying the
dissertation requirement for the degree of Doctor of Philosophy.

Recommended to the Graduate Council

Date_____ _____

Sherief Reda, Advisor

Date_____ _____

Jacob Rosenstein, Reader

Date_____ _____

Ayse K. Coskun, Reader

Date_____ _____

Adel Belouchrani, Reader

Approved by the Graduate Council

Date_____ _____

Andrew G. Campbell, Dean of the Graduate School

Vitae

Sofiane Chetoui was born in Tebessa, Algeria. He received his State Engineering Degree and M.Sc. in Electronics from Ecole Nationale Polytechnique, Algeria in 2017. He received his M.Sc. in Electrical and Computer Engineering from Brown University in 2020 during his studies in the Ph.D. program. His research aims to improve thermal behavior, energy efficiency and performance of mobile devices through runtime management. He is interested in thermal/power sensing, hardware characterization and modeling of modern Mobile SoCs. He uses control theory and machine learning to design the next generation of thermal and power management techniques. His research focuses as well on providing a better user-experience and battery lifetime management for Mobile Devices.

sofiane_chetoui@brown.edu

Brown University, RI, USA

Selected Publications:

1. S. Chetoui, M. Chen, A. Golas, F. Hijaz, A. Belouchrani, S. Reda, "Alternating Blind Identification of Power Sources for Mobile SoC", in Proceedings of the ACM/SPEC International Conference on Performance Engineering 2022.

2. S. Chetoui, R. Shahi, S. Abdelaziz, A. Golas, F. Hijaz, S. Reda, "ARBench: Augmented Reality Benchmark For Mobile Devices", under revision in IEEE Interna-

tional Symposium on Performance Analysis of Systems and Software (ISPASS) 2022.

3. S. Chetoui, S. Reda, ”Coordinated Self-tuning Thermal Management Controller for Mobile Devices”, in IEEE Design Test. 2020 Feb 28;37(5):34-41.

4. S. Chetoui, S. Reda, ”CasCon: Cascaded Thermal And Electrical Current Throttling for Mobile Devices”, in IEEE Embedded Systems Letters. 2021 May 13.

5. S. Chetoui, S. Reda, ”Workload-and User-aware Battery Lifetime Management for Mobile SoCs”, in 2021 IEEE Design, Automation Test in Europe Conference Exhibition (DATE) 2021 Feb 1 (pp. 1679-1684).

6. M. Said, S. Chetoui, A. Belouchrani, S. Reda, ”Understanding the sources of power consumption in mobile SoCs”, in 2018 Ninth International Green and Sustainable Computing Conference (IGSC) 2018 Oct 22 (pp. 1-7). IEEE.

7. Z. Yuan, P. Shukla, S. Chetoui, S. Nemtzow, S. Reda, AK. Coskun, ”PACT: An extensible parallel thermal simulator for emerging integration and cooling technologies”, in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. 2021 May 11.

8. Z. Yuan, P. Shukla, S. Chetoui, C. Knox, S. Nemtzow, S. Reda, AK. Coskun, ”Towards Fast and Accurate Parallel Chip Thermal Simulations with PACT”.

Acknowledgements

This thesis would not have been possible without the constant support, guidance and inspirations of many kind individuals. First and foremost, I would like to express my immense gratitude to my advisor and mentor, Prof. Sherief Reda, whose guidance, support, and valuable insights during the course of my research has made this thesis possible.

I would also like to thank Prof. Jacob Rosenstein, Prof. Ayse K. Coskun and Prof. Adel Belouchrani for being on my defense committee and taking the time to review my thesis.

I am extremely thankful for the productive collaborations with all my co-authors, Prof. Adel Belouchrani, Prof. Ayse K. Coskun, Prof. Mostafa Said, Dr. Zihao Yuan, Abhinav Golas, Farrukh Hijaz, Andey Donzelli, Michael Chen, Rahul Shahi, my advisor Prof. Sherief Reda, and many others. My work would not have been possible without them. Specifically, I would like to sincerely thank Prof. Adel Belouchrani for his valuable insights and guidance.

I would like to thank Samsung SARC for giving me internship opportunities twice. Especially, I would like to thank Can Hankendi for his distinguished mentorship. I would like to thank the Power Methodology team at Nvidia for giving me an amazing internship opportunity, more specifically, I would like to thank Kapil Dev for his distinguished mentorship and guidance.

I would also like to thank my fellow graduate students and friends at Prof. Reda's group at Brown: Reza Azimi, Soheil Hashimi, Hokchhay Tran, Abdelrahman Hosny, Farnaz Nouraei, Marina Hesham, Jingxiao Ma, Ahmed Agiza and many others for always making the last five years memorable.

I would like to express my immense gratitude to Ghania Rais for her great care and for being a one-of-a-kind teacher to me. I am greatly thankful for Mohamed Saoudi for his advice and support during and after my high school studies. I would like to express my deep gratitude for Noureddine Tayebi for his support and help before and during my PhD track.

I would like to thank Hala Lemmouchi for the years of support, encouragements, and many unforgettable memories.

Last but not least, I would like to thank my parents, M. Larbi Chetoui and Leila Chamekh, and my siblings for their unwavering support and love. I have learned a lot from them. Without them, none of what I achieved today would be possible.

.

Abstract of "Thermal and Power Sensing and Management for Mobile System-On-a-Chip" by Sofiane Chetoui, Ph.D., Brown University, April 2022

Mobile devices became an essential part of daily life with the increased computing capabilities and features. The leaps that Mobile System-On-a-Chip (SoC) have taken in the past decade led to an explosive growth of mobile devices. Mobile SoCs have become the leading product in the semiconductor industry, due to its continually improving performance and decreasing cost. Their designs have rapid evolution, at least a new design is released each year, and new designs tend overshadow the older ones. Technology scaling, heterogeneous multiprocessor designs and the integration of specialized hardware units are among the main factors enabling the performance increase. This increase has enabled new applications that require intensive computation, raise critical thermal, power and energy constraints. Elevated chip temperatures and power density substantially affect the user experience due to performance throttling and decreased battery lifetime.

This thesis aims at improving the performance and the battery lifetime of mobile devices through thermal and power sensing and management, under thermal, power and energy constraints. On the sensing side, this thesis first introduces an Alternating Blind Identification of Power sources (Alternating-BPI), a technique that accurately estimates the power consumption of individual SoC units without the use of any design based models. The proposed technique uses a novel approach to blindly identify the sources of power consumption, by relying only on the measurements from the embedded thermal sensors and the total power consumption. Additionally, on the sensing side we propose a Deconvolutional Neural Network (DCNN) based power map estimation. The proposed approach relies only on the usage of the few available embedded thermal sensors on the SoC to estimate the full SoC power map. Afterwards, the thesis tries to take a step towards the improvement of the Augmented Reality mobile experience by performing a power and hardware characterization of Augmented Reality Applications. The characterization includes designing and developing ARBench, an augmented reality benchmark for mobile

devices. The benchmark and the Alternating-BPI technique were used to study the performance and power trade-offs of different CPU multi-core configurations, in order to provide insights to save power while meeting the AR performance requirements. On the runtime management side, this thesis proposes a coordinated self-tuning thermal management controller on the basis of online deep learning that continuously adapts to characteristics and operating conditions, while taking into account both skin temperature and junction temperature constraints in a coordinated manner. We also propose a cascaded controller for mobile devices that controls the different sources of current and thermal emergencies in a coordinated manner. The controller design was inspired from the physical relations that exist among the different current and thermal measures, and it allows to achieve better performance while saving power. Finally, this thesis introduces a novel workload- and user-aware battery lifetime management technique that maximizes the performance under the user's desired battery lifetime. Our approach leverages insights about the running workloads by collecting CPU-GPU performance counters, which are used to proactively scale the CPU-GPU frequencies using machine learning.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Problem Characterization

Mobile devices have become an indispensable tool of daily life, with 6.64 billion smartphone users across the globe. The evolution of mobile devices towards multi-purpose portable devices, used for communication, computing and entertainment is the main reason behind their widespread. Mobile usage is expected to keep growing in the future, as mobile technologies are becoming more affordable and available than ever. The increasing computing capabilities of mobile devices enabled them to replace desktop computers in many tasks, and with the advent of new technologies like Virtual and Augmented Reality, mobile devices are taking an even more important role.

The growth in the number of transistors per silicon die has been one of the main driving forces in the improvement of the computing capabilities of Mobile SoCs. Moore's law predicted that the number of transistors in a dense integrated circuit doubles about every two years. The slowing of Moore's law has led the industry to explore other direc-

tions, such as the design of heterogeneous multi-processor designs and the integration of specialized hardware units.

The combination of these different technologies enabled new applications and features that require intensive computation, raising critical thermal, power, and energy constraints:

- **Thermal and power constraints**: As technology scales down, power density considerably increases, raising critical power and thermal challenges. This is even more challenging for mobile devices due to their form factor, the multiple sources of thermal emergencies, their limited cooling capabilities, as well as the limited electric current delivery, arising from battery specifications. Elevated chip temperature triggers performance throttling mechanisms, making such temperature levels a major performance bottleneck [73, 58, 83]. Preventing thermal violations while sustaining the performance at acceptable levels led thermal management to become a critical factor in determining the user experience. The temperature limits are usually defined by the junction temperature of the chips, which has to be maintained usually under 90 °C. The other temperature that defines the limits for mobile devices is the skin temperature, which has to be maintained under 40 °C; otherwise, the mobile device would create a burning sensation to the end user [23, 29, 32, 41]. Additionally, the electric current should be sustained at safe levels to respect the battery discharge rate and the deterioration of the battery cells [85, 38, 45]. As mobile devices are battery-powered, there are hardware and software mechanisms that keep the current within the battery discharge specifications. In case the current exceeds some predefined value, the software mechanism throttles the frequency, with a fall back to the hardware mechanism, which shuts down the phone for safety reasons. Thus, it is essential to devise runtime management techniques that can prevent thermal and power violations, while maximizing the performance.

- **Energy constraints**: Battery lifetime has become one of the top usability concerns of mobile systems [81]. The phone form factors impose a strict limit to the battery size that can be accommodated. Thus, mobile devices are highly battery constrained because users are implicitly expected to charge their mobile devices once a day. On the other hand, the functionality integrated into such devices, and consequently their power consumption will continue to grow. Various studies [51, 40, 53] tried to build systems that adaptively balance performance and battery lifetime, since the user experience is mainly determined by these two factors. Thus developing energy management techniques to provide an extended battery lifetime, while meeting the user performance expectation is critical for the user experience.

1.2 Major Thesis Contributions

In this section, we outline the major contributions made in this thesis regarding the exploration of new techniques of thermal and power sensing and management for mobile system-on-a-chip.

1. **Alternating Blind Identification of Power Sources for Mobile SoCs:** In Chapter 3, we introduce Alternating Blind Identification of Power sources (Alternating-BPI), a technique that accurately estimates the power consumption of individual SoC units without the use of any design based models. The proposed technique uses a novel approach to blindly identify the sources of power consumption, by relying only on the measurements from the embedded thermal sensors and the total power consumption. The accuracy and applicability of the proposed technique was verified using simulation and experimental data. Alternating-BPI is able to estimate the power at the SoC hardware unit level with up to 98.1% accuracy. Furthermore,

we demonstrate the applicability of the proposed technique on a commercial SoC and provide a fine-grain analysis of the power profiles of CPU and GPU Apps, as well as Artificial Intelligence (AI), Virtual Reality (VR) and Augmented Reality (AR) Apps. To summarize, the contributions of this chapter are as follows:

- We introduce the first Alternating Blind Identification of Power sources (Alternating-BPI). The new approach allows a better accuracy and practicality than previous blind identification techniques, and works on both simulation and experimental data, as it does not require steady thermal states.
- The proposed technique substantially decreases the power estimation error, especially for heterogeneous SoCs with multiple hardware units. Simulation data has shown that the proposed technique decreases the power estimation error as low as 1.9%, as compared to 11.2% for BPI [66]. Furthermore, we show that the accuracy of the proposed technique remains stable when moving from homogeneous to heterogeneous architectures, and remains stable when the number of hardware units increase. As opposed to BPI [66] and BPISS [72], whose accuracy dropped when increasing the number of units and moving to heterogeneous architectures.
- The technique is used to design a plug and play tool, that is made publicly available [16], and that allows to estimate the power consumption of SoC units.
- The proposed technique is demonstrated using simulated and experimental data. Then it is used to characterize the power profile of several benchmarking Apps on a commercial SoC, including : CPU, GPU, Artificial Intelligence (AI) , Virtual Reality (VR), Augmented Reality (AR) Apps. The power characterization provides insights about the power efficiency of the different hardware units on a state-of-the-art commercial SoC.

2. **Deconvolutional Neural Network Based Power Map Estimation:** In chapter 4, we propose an approach for full-chip power map estimation based on Deconvolutional Neural Networks (DCNN). The proposed approach relies on the usage of the few available embedded thermal sensors on the SoC to estimate the full SoC power map. The contributions of this chapter are as follows:

    - We propose to solve the power map estimation problem as an image generation problem using Deconvolutional Neural Networks (DCNN). The proposed DCNN takes as input the thermal measurements from the embedded thermal sensors and the total power to estimate the full SoC power map.
    - The proposed technique allows to estimate the power map at a finer spatial granularity than the thermal spatial granularity of the existing thermal sensors. More specifically, it allows to estimate the power even at locations where thermal measurements are not physically available.
    - The proposed technique is demonstrated using a commercial SoC while running several benchmarks. The predicted power maps show a 97% similarity (2D correlation) with the power maps estimated using the Alternating-BPI.

3. **Power and Hardware Characterization of Augmented Reality Applications:** In Chapter 5, we take a step towards the improvement of the Augmented Reality mobile experience by performing a power and hardware characterization of Augmented Reality Applications. The characterization includes designing and developing AR-Bench, an augmented reality benchmark for mobile devices. We provide an analysis of the existing AR Apps in terms of performance, power and hardware utilization to motivate the design of ARBench. The proposed benchmark includes various workloads, such that each workload evaluates a particular aspect of AR workloads by stressing multiple hardware units of the SoC (CPU, GPU, DSP, etc). Then, AR-Bench is used to evaluate the AR performance of existing commercial mobile de-

vices and Android operating systems. Furthermore, we use ARBench to perform a phase analysis to identify a set of canonical phases that could be used to model AR workloads, and use them to characterize existing AR Apps. Finally, the benchmark is used to study the performance and power trade-offs of different CPU multi-core configurations, and we provide insights that could be used to save power while meeting the AR performance requirements. To summarize, the contributions of this chapter are as follows:

- We characterize the hardware utilization of several AR Apps using a commercial device. Then, we analyze the power consumption per-hardware unit and per-AR process.
- While existing AR benchmarks mainly target Desktop computers, we design and develop ARBench, the first AR benchmark that measures the AR performance of mobile devices. The benchmark incorporates different AR workloads that stress multiple hardware units of the SoC (e.g. CPU, GPU, DSP), and measures the individual score for each AR workload.
- We provide insights about the ability of existing mobile devices to run AR workloads, by using ARBench to evaluate the performance of several commercial mobile devices and Android Operating Systems. The performance results are shown for each individual AR workload.
- We use ARBench to perform a phase analysis to identify a set of canonical phases that could be used to model and characterize AR workloads.
- We study the performance and power trade-off of different multi-core CPU configurations, and provide insights about the most efficient multi-core configuration that could run the AR workloads, while meeting the performance requirements.

4. **Coordinated Self-tuning Thermal Management Controller for Mobile Devices:** In Chapter 6, we propose a coordinated self-tuning thermal management controller on the basis of online deep learning that continuously adapts to characteristics and operating conditions. Furthermore, our controller takes into account both skin temperature and junction temperature constraints in a coordinated manner. To summarize, the contributions of this chapter are as follows:

    - We design a junction temperature controller that continuously tunes the proportional–integral–derivative (PID) parameters on the basis of online learning. The controller uses a neural network (NN) that updates its weights according to the operating conditions to reduce thermal violations while maximizing performance.
    - We design an NN-based coordinated self-tuning thermal management controller that manages both the skin and the junction temperature by proactively scaling the junction temperature threshold.
    - We implemented a low-overhead controller on a real smartphone and we evaluate it comprehensively compared to PID [84], thermal-aware DVFS controller [48] and USTA [41] , under different ambient temperatures and workload characteristics. Our results demonstrate that our coordinated self-tuning thermal controller leads to 6% better performance, and spends up to 27$\times$ less time in thermal violation.

5. **CasCon: Cascaded Thermal And Electrical Current Throttling for Mobile Devices:** In Chapter 7, we propose a cascaded thermal and electrical current throttling controller inspired from the physical relations that exist among the different current and thermal measures. The proposed controller achieves better results by coordinating between the different sources of current and thermal emergencies, and dynamically adjusting their caps. Testing on a state-of-the-art smartphone, the proposed technique achieves 6.5% better performance and 18% power savings as compared to existing techniques, while avoiding current and thermal violations using 40× less DVFS transitions. To summarize, the contributions of this chapter are as follows:

   - We design a cascaded controller (CasCon) that manages the skin temperature, the junction temperature and the electrical current in a coordinated manner. In contrast to existing work, where the skin and junction temperature, and the electrical current are managed separately, leading to a sub-optimal control.
   - In contrast to existing work, the proposed controller dynamically changes the thermal and electrical current caps in runtime, allowing it to save power and improve performance. We also introduce a frequency locking function that significantly reduces the number of DVFS transitions, hence bringing extra power savings.
   - We implement our CasCon controller on a real smartphone and we evaluate it comprehensively at different ambient temperatures. Our results show that the proposed controller successfully prevents current and thermal violations even at high ambient temperatures, while bringing up to 6.5% performance improvements and 18% power savings compared to previous work.

6. **Workload- and User-aware Battery Lifetime Management for Mobile SoCs:** In Chapter 8, we propose a CPU-GPU workload- and user- aware battery lifetime management technique for mobile devices using machine learning. Firstly, we design a workload-aware governor through an offline and an online analysis. A set of CPU and GPU performance counters is used during the offline analysis to identify a set of canonical phases (CP). In runtime, k-means is used to classify each sample of the performance counters to one of the predefined CP. Afterwards, we build a model that predicts the energy consumption given the user usage history. Finally, the energy model is used to find the optimal frequency settings for the CPU and GPU to provide the best performance while meeting the target battery lifetime. The evaluation of the proposed work against state of the art techniques in a commercial smartphone, shows 15.8% and 9.4% performance improvement on the CPU and GPU, respectively. The proposed technique also shows 10× improvement in performance variation, while meeting the desired battery lifetime. To summarize, the contributions of this chapter are as follows:

    - We design the first workload- and user-aware battery lifetime management technique. The workload-awareness is achieved through performance counters, while the user awareness is incorporated by representing the user-usage history through a set of canonical phases (CP).
    - We propose a novel model that predicts the energy consumption based on the user usage history. This model makes the proposed technique user-aware, and helps in better meeting the user desired lifetime.
    - The proposed battery lifetime management is achieved by scaling both the CPU and the GPU DVFS levels, unlike previous techniques that do not consider the GPU.
    - We implement our technique on a commercial smartphone and compare its

performance against state-of-the-art battery management techniques. We show that our technique achieves 15.8% and 9.4% performance improvement on the CPU and GPU, respectively, while meeting the lifetime target and decreasing the performance variation by 10x.

The organization for the remainder of this thesis is as follows. Chapter 2 briefly describes the background and related prior work. Next, Chapter 3 presents our proposed Alternating Blind Identification of Power source technique. Chapter 4 introduces the Deconvolutional Neural Networks based power map estimation. Chapter 5 shows the results of the power and hardware characterization of Augmented Reality Applications. Next, we introduce our proposed coordinated self-tuning thermal management controller in Chapter 6. In Chapter 7, we propose CasCon, a cascaded thermal and electrical current throttling controller for mobile devices. Chapter 8 introduces a novel workload- and user-aware battery lifetime management technique. Finally, Chapter 9 summarizes the results and findings of this thesis and provides possible future extensions of this work.

Chapter 2

Background

In this chapter, we describe the background and related prior work to the proposed techniques in this dissertation. We start by introducing thermal and power sensing background for Mobile SoCs in Section 2.1. In Section 2.2, we introduce related work to thermal and power runtime management. Finally, in Section 2.3 we introduce related work to energy and battery lifetime management.

2.1 Thermal and power sensing

The ability to measure power consumption of different hardware units is essential for the operation and improvement of mobile SoCs, as well as the enhancement of the power efficiency of the software that runs on them. Mobile SoCs are usually enabled with embedded thermal sensors to measure the temperature at the hardware unit level; however, they lack the ability to sense the power. Thus, devising techniques that enable fine-grain level power profiling of Mobile SoCs, and the software that runs on them, is a major step towards im-

plementing efficient power and thermal management techniques, as well as, designing the architecture of the next generation Mobile SoCs.

Various studies investigated a wide range of methods for thermal and power modeling of heterogenuous SoCs. The standard approach used by these techniques try to identify the state space model that links temperature to power [66, 72, 25, 35, 78] :

$$\mathbf{t}(k) = \mathbf{A}\mathbf{t}(k-1) + \mathbf{B}\mathbf{p}(k), \tag{2.1}$$

where $\mathbf{t}$(k) an $\mathbf{p}$(k) are vectors that denote the temperature and power of the SoC hardware units at time k, respectively. Both $\mathbf{A}$ and $\mathbf{B}$ are two modeling matrices that capture the physical relationship between power and temperature. More precisely, the $\mathbf{A}$ matrix represents the thermal conductance matrix, which describes the natural response of the system, in the absence of power. The $\mathbf{B}$ matrix represents the forced response matrix, and it is function of the thermal capacitance and the thermal conductance matrices. Both the $\mathbf{A}$ and $\mathbf{B}$ are square matrices, whose dimension is equal to the number of power sources. The state space model in Equation 2.1 is derived from the heat diffusion equation [56], which describes the power-thermal interaction by taking into consideration the thermal conductivity, the density and the specific heat of the material. More specifically, the derivation of the model in Equation 2.1 is performed by applying a spatial discretization on the heat diffusion equation [56] , followed by a temporal discretization.

Being able to compute the state space modeling matrices would make it possible to estimate and predict the power consumption, at the same level of granularity as the available thermal measurements. In the case where thermal measurements are unavailable, and power measurements are available, being able to compute the modeling matrices would make it possible to predict the temperature, at the same level of granularity as the avail-

able power measurements. There are three general approaches to identify the state space model:

1. **Design based approach**: it is usually based on pre-silicon data and requires access to the design proprietary information, such as its layout and gate level netlist, its materials and heat sink configuration [47, 28, 57]. Thus, this approach assumes the availability of $\mathbf{A}$ and $\mathbf{B}$ and attempts to estimate $\mathbf{p}$(k). Skadron *et al.* [77] designed an approach that models thermal behavior of the die and its package as a circuit of thermal resistances and capacitances, that correspond to functional blocks at the architecture level, which helps in capturing the physical relationship that relates power to temperature, which is similar to finding the $\mathbf{A}$ and $\mathbf{B}$ matrices. However, this is not a practical solution, since it is design specific, and it is prone to errors related to variability in the semiconductor manufacturing process. Additionally, it is computationally challenging to conduct large-scale gate-level simulation.

2. **Runtime based approach**: it considers the processor as a gray box, and it identifies the state-space models based on physical measurements. Most of these techniques use system identification techniques and mainly rely on the existence of sensors for the power sources [25, 52, 27]. Thus, this approach assumes the availability of $\mathbf{p}$(k) and attempts to estimate $\mathbf{A}$ and $\mathbf{B}$. However, such fine-grain power sensors are unavailable on mobile SoCs. Another type of runtime approach, relies on the usage of infrared imaging and performance counter measurements [68, 39], however, performing infrared imaging might be invasive and prone to noisy measurements. Other runtime based techniques focus on the usage of performance counters to model the power, for instance, Min *et al.* [54] proposed a general approach to build system power estimation models based on hardware performance counters. Karan *et al.* [75] derive functions for real time estimation of system power consumption using performance counters. Kim *et al.* [49] proposed a statistical approach for build-

ing power models using performance counters as effective proxies for x86 systems. However, all the previous techniques assume the existence of power sensors and do not perform their power modeling at the fine-grain level.

3. **The blind identification approach**: it makes no assumption about the availability of modeling matrices and the fine-grain power sensors. This approach relies on the total power and the fine-grain thermal sensors to estimate both the modeling matrices and the power sources [66]. This technique relies on the steady state thermal data to estimate the $\mathbf{B}$ matrix. However, usually in practice it is not possible to reach steady thermal state on modern SoCs. This affects the accuracy of the model and makes it less convenient to use in practice.

The challenging task of getting fine-grain power measurements has led to the existence of only few studies that provide useful insights about the power consumption and efficiency of the different SoC hardware units while stressed by various software applications [59, 30, 72].

2.2 Thermal and power runtime management

Mobile devices are battery-powered and composed of various computing units with a very constrained form factor. More specifically, mobile devices should stay small enough to hold and operate in the hand. As a result, only limited cooling techniques can be used. The increasing user demand and the emerging new technologies are pushing the power density of System-on-chips (SoCs) to new boundaries. This increase leads to more power and thermal challenges, which have been investigated in various studies:

- **Junction temperature management:** Form-factor constraints make thermal management on mobile devices a unique challenge as compared to thermal management on desktop or server processors. For instance, mobile SoCs have much faster thermal transient than desktop processors since the thermal capacitance of their cooling systems is significantly lower. Elevated chip temperature triggers performance throttling mechanisms, making such temperature levels a major performance bottleneck. Thermal modeling and predictive models have been widely investigated. In particular, Gaurav *et al.* [76] used a power and thermal model to predict temperature to dynamically compute a power budget based on which the frequency is determined. Prakash *et al.* [64] proposed CPU-GPU coordinated thermal management using two separated PID controllers and thermal prediction. A RC model that reflects the thermal coupling between the battery and the application processor is proposed by Xie *et al.* [79]; the model is then used to predict the temperature and pre-compute safe frequencies online. Achieving sustainable performance by compromising short term performance was investigated by Sahin *et al.* in [70] and [71], where a quality-aware frequency scaling is proposed to improve performance sustainability. Different tracks in thermal management were also investigated. Bartolini *et al.* [26] designed energy aware thermal controller, based on prediction models. Kim *et al.* in [48] proposed a thermal-aware DVFS scaling scheme for mobile devices that stabilizes the frequency by averaging over a moving window. Cochran *et al.* [36] proposed thermal prediction and adaptive control phase detection based on performance counters. Even if various elaborated techniques have been proposed in the literature, in commercial smartphone the most conventionally used thermal management controller is the PID, whose overall control function can be expressed mathematically as:

$$u(t) = K_p e(t) + K_i \int_0^t e(t')dt' + K_d \frac{de(t)}{dt}, \tag{2.2}$$

where K_p is the proportional gain, K_i is the integral coefficient and K_d is the derivative coefficient, while $u(t)$ and $e(t)$ represent the control signal and the error between the current maximum temperature and the thermal threshold of the CPU, respectively. The tuning of the PID parameters determines the performance of the controller. The mostly used tuning method is Zigler-Nichols [84].

Zigler-Nichols [84] is a heuristic method based on a closed-loop system, where the K_i and K_d gains are first set to zero. The K_p gain is then increased (from zero) until it reaches the critical gain K_{cr}, at which the output of the control loop has sustained oscillations with period P_{cr}. Finally, K_{cr} and the oscillation period P_{cr} are used to set the K_p, K_i, and K_d gains, where $K_p = 0.6K_{cr}$ and $T_i = 0.5P_{cr}$, and $T_d = 0.125P_{cr}$. Once the oscillatory settings are dialed back according to the Zigler-Nichols coefficients (i.e., 0.6, 0.5, and 0.125), the controller no longer leads to a sustained thermal oscillation state, but rather a stable thermal behaviour.

- **Skin temperature control:** Another challenge about mobile devices is that they are handheld; thus, the whole device's temperature has to be limited because of skin temperature [44]; otherwise, it would adversely impact the user's experience. It has been shown that the skin temperature could be a performance limiter [79, 55, 60, 41]. The authors [79] showed that the performance is linearly related to the skin temperature. Park *et al.* [60] conducted experiments to explain the relationship between the operating characteristics and heat generation for all active components in a smartphone device, then they developed a thermal prediction model that accurately predicts the skin temperature of a mobile device. Skin temperature management has not been as widely investigated as the junction temperature management, however a User−specific skin temperature-aware (USTA) is proposed in [41]. The technique relies on a machine learning model to predict the skin temperature, which is derived offline based on empirical data gathered from thermal measurements.

- **Current throttling:** As mobile devices are battery-powered, the electric current should be sustained at safe levels to respect the battery discharge rate and the deterioration of the battery cells [82]. There are hardware and software mechanisms that keep the current within the battery discharge specifications. In case the current exceeds some predefined value, the software mechanism throttles the frequency, with a fall back to the hardware mechanism, which shuts down the phone for safety reasons.

2.3 Energy and battery lifetime management

Battery lifetime has become one of the top usability concerns of mobile systems [81]. The user experience, for battery powered-devices, depends on both performance and battery lifetime.

Various studies tried to offer better energy savings by studying DVFS governors. In particular, a hierarchical FSM-based frequency capping technique was proposed to save power by allowing minor degradation in performance [61]. Choi *et al.* designed a graphics-aware power governing technique that solves the energy inefficiency of frame rendering [33]. A memory-aware cooperative CPU-GPU DVFS governor was proposed to maximize the energy efficiency while meeting a performance target [46]. A phase-aware web browser power management technique was proposed by Peters *et al.* [62]. In an other study, an energy-efficient mobile web interaction framework that leverages a cloud-based machine learning model was proposed [80]. Most of these techniques allow a slight degradation in performance to improve the energy efficiency, as result, they do not properly offer a fine-grain balancing of the performance and the battery lifetime. Furthermore, they do not take into consideration the user's desired battery lifetime goal.

The most used technique in commercial smartphones for battery lifetime management is Powersave [42]. It sets the allowed frequency to the highest possible level, and when 20% of the battery capacity is reached, the allowed frequency is set to a lower level.

Prior studies have also explored improving the user satisfaction by balancing the battery lifetime and the performance. Yan *et al.* [81] proposed a quality of experience (QoE)-aware frequency governor which dynamically scales the CPU frequency at low battery levels. Donohoo *et al.* [40] proposed a framework that optimizes the CPU and the screen backlight energy consumption. Poyraz *et al.* [63] used built-in sensors to predict the user satisfaction for CPU Settings to save energy. In a recent study, Lee *et al.* [51] proposed BUSQ1 and BUSQ3, two dynamic quality of service (QoS) scaling approaches that automatically balance QoS and energy. BUSQ1 defines the discharge profile linearly, while BUSQ3 defines it based on the user usage history. Afterwards, by comparing the current battery status against the predefined discharge profile, the frequency is decreased if energy savings are needed and is increased otherwise. However, the previously mentioned studies do not take into consideration the GPU. Furthermore, most of these studies do not leverage any workload-awareness, rather, they scale the frequency independently of the running workload. Thus, wasting several opportunities to save energy or to improve performance.

Chapter 3

Alternating Blind Identification of Power Sources for Mobile SoCs

In this chapter, we investigate the use of a new approach that relies on an Alternating-BPI algorithm to blindly estimate the power at the SoC unit level. We use the proposed technique to develop a plug and play tool that allows to identify the power consumption of the different SoC units. Using the proposed technique, we provide a fine-grain power analysis of a commercial SoC and provide useful insights about its power efficiency. The contributions of this chapter are as follows:

- We introduce the first Alternating Blind Identification of Power sources (Alternating-BPI). The new approach allows a better accuracy and practicality than previous blind identification techniques, and works on both simulation and experimental data, as it does not require steady thermal states.
- The proposed technique substantially decreases the power estimation error, especially for heterogeneous SoCs with multiple hardware units. Simulation data has

shown that the proposed technique decreases the power estimation error as low as 1.9%, as compared to 11.2% for BPI [66]. Furthermore, we show that the accuracy of the proposed technique remains stable when moving from homogeneous to heterogeneous architectures, and remains stable when the number of hardware units increase. As opposed to BPI [66] and BPISS [72], whose accuracy dropped when increasing the number of units and moving to heterogeneous architectures.

- The proposed technique is demonstrated using simulated and experimental data. Then it is used to characterize the power profile of several benchmarking Apps on a commercial SoC, including : CPU, GPU, Artificial Intelligence (AI) , Virtual Reality (VR), Augmented Reality (AR) Apps. The power characterization provides insights about the the power efficiency of the different hardware units on a state-of-the-art commercial SoC. Some of the insights include: (1) Even with the new integrated computing units to modern Mobile SoCs, like the GPU, the image signal processor and neural engine, the CPU is still the main source of power consumption, representing around 60% to 75% of the total SoC power. (2) The little CPU cluster plays a major role in saving power, with a power consumption that is 5x less than the big CPU cluster. (3) The GPU power consumption for AR Apps, represents only 9% of the total SoC power consumption, while the CPU represents 75% of the total SoC power consumption.

The rest of the chapter is organized as follows: Section 3.1 motivates the proposed work. Section 3.2 describes the proposed technique and the underlying physical and mathematical concepts, and it introduces the Alternating-BPI tool. Section 3.3 presents the evaluation results of our technique compared against state-of-the-art techniques, as well as the power characterization of different benchmarking Apps on a commercial SoC. Section 3.4 concludes the chapter.

3.1 Motivation

The higher power density and limited cooling solutions for mobile SoCs is pushing mobile devices towards a major performance bottleneck [44]. Thus, devising techniques and tools that help in profiling at a fine-grain level the existing SoCs, and the software that runs on SoCs, is a major step towards implementing efficient power and thermal management techniques, as well as, designing the architecture of the next generation Mobile SoCs.

Sensing the power and temperatures of the different hardware units in modern SoCs is a key enabling efficient and optimal power and thermal management techniques. Additionally, a fine-grain map of the power consumption of modern SoCs across different software applications would provide valuable insights to improve the performance of SoCs at the hardware and software level. However, modern processors provide only coarse-grain power measurements of all cores using the running average power limit (RAPL) [37]. Various techniques from the literature rely on different assumptions and offer different levels of accuracy, practicality, and feasibility have been proposed [28, 57, 27, 39].

The availability of thermal sensors on a per-hardware unit basis on mobile SoCs, makes it possible to estimate the power consumption at the unit level using blind identification techniques. Blind Power Identification (BPI) [66, 67, 72] was proposed to estimate the power at the SoC unit level, based on the individual thermal measurements and the total power consumption. However, BPI [66] relies on the thermal steady state data to estimate the power model parameters, which is hard to generate in practice and might affect the accuracy. The reliance on certain tools and assumptions, the insufficient accuracy and the lack of practicality of the available power identification techniques has led to the lack of techniques that provide a fine-grain power measurements of modern SoCs.

3.2 Alternating Blind Identification of Power sources

3.2.1 The proposed approach

The proposed approach consists of identifying the state space modelling matrices with a better accuracy. The state space model given by:

$$\mathbf{t}(k) = \mathbf{A}\mathbf{t}(k-1) + \mathbf{B}\mathbf{p}(k), \tag{3.1}$$

where $\mathbf{t}$(k) an $\mathbf{p}$(k) are vectors that denote the temperature and power of the SoC hardware units at time k, respectively. Both $\mathbf{A}$ and $\mathbf{B}$ are two modeling matrices that capture the physical relationship between power and temperature. More precisely, the $\mathbf{A}$ matrix represents the thermal conductance matrix, which describes the natural response of the system, in the absence of power. The $\mathbf{B}$ matrix represents the forced response matrix, and it is function of the thermal capacitance and the thermal conductance matrices. Both the $\mathbf{A}$ and $\mathbf{B}$ are square matrices, whose dimension is equal to the number of power sources.

The proposed technique consists of two steps: *Off-line Training* step and *Runtime Estimation* step. The *Off-line Training* step needs to be run only once for each SoC, to capture the modeling matrices. The *Runtime Estimation* needs to be run each time the per-SoC unit power needs to be identified, its little computation overhead allows it to run on the device in runtime.

In the *Off-line Training* step a training data is required as input. It consists of the per-unit SoC thermal measurements and the total power consumption. The thermal measurements are generated while stressing the SoC hardware units using different patterns

separated by sleep periods. More precisely, different combination of stressed and idle units should be part of the training data. This allows to capture thermal transients, and the contribution of the different units to the total power. The output of the *Off-line Training* is the state space model matrices $\mathbf{A}$ and $\mathbf{B}$. On Algorithm 1, the *Off-line Training* step is shown through lines 1 to 5, which shows how the $\mathbf{A}$ and $\mathbf{B}$ matrices of Equation 3.1 are identified.

First, the natural response matrix is estimated using the transient temperature traces. After stressing the cores with a workload, the power is forced to $\mathbf{p}$(k)=0. In practice, this could be achieved by stopping the running workload, and turning-off the target hardware units. Thus from Equation 3.1, we get:

$$\mathbf{t}(k) = \mathbf{A}\mathbf{t}(k-1), \tag{3.2}$$

Equation 3.2 is used to determine the $\mathbf{A}$ matrix through least square minimization, as shown in line 1 of Algorithm 1. This minimization is solved under the positivity constraint of the A matrix, since the A matrix represents the thermal conductance matrix, as explained earlier in Chapter 2.

Next, the goal is to make an initial guess about the $\mathbf{B}$ matrix. During a steady temperature state, there is no thermal variation, so the temperature at time k would be equal to temperature at time $k-1$, i.e. $\mathbf{t}(k) \approx \mathbf{t}(k-1) = \mathbf{t}_s$, which gives the following using Equation 3.1:

$$\mathbf{t}_s \approx \mathbf{A}\mathbf{t}_s + \mathbf{B}\mathbf{p}_s, \tag{3.3}$$

Algorithm 1: Alternating Blind Identification of Power Sources.

Input: Temperatures $\mathbf{t}$(k) , Total power of $\mathbf{p}$(k)

Output: Natural Response Matrix $\mathbf{A}$, Forced Response Matrix $\mathbf{B}$, Power Profiles $\mathbf{p}$(k)

1 Find the Natural Response Matrix $\mathbf{A}$ through the least square minimization:

$$\min |\mathbf{t}(k) - \mathbf{A}\mathbf{t}(k-1)|^2$$

under the constraint $\mathbf{A} \succeq 0$

2 Initialize the $\mathbf{B}$ matrix :

$$\mathbf{R} = (\mathbf{J} + \mathbf{I})/3$$

$$\mathbf{B} = (\mathbf{I} - \mathbf{A})\mathbf{R}$$

where I is the identity matrix, J is an all ones matrix, and R is the thermal transfer matrix

3 **Repeat** **P**ower and **B**-Matrix estimation steps for n times:

4 $\mathbf{P}$-*step*: Using quadratic programming find $\mathbf{p}$(k) $\succeq 0$ such that:

$$\min \mathbf{B}\mathbf{p}(k) - (\mathbf{t}(k) - \mathbf{A}\mathbf{t}(k-1))_2$$

$$\mathbf{p}(k)_1 = \mathbf{p_{tot}}(\mathbf{k})$$

where $\mathbf{p_{tot}}(\mathbf{k})$ is the measured total power at time k

5 $\mathbf{B}$-*step*: Using least square minimization find $\mathbf{B}$ given $\mathbf{p}$(k) and $\mathbf{A}$:

$$\min |\mathbf{B}\mathbf{p}(k) - (\mathbf{t}(k) - \mathbf{A}\mathbf{t}(k-1))|^2$$

6 **Runtime estimation:** Given the $\mathbf{A}$ and $\mathbf{B}$, and the target thermal and total power data, solve the quadratic programming of the $\mathbf{P}$ step.

$$\mathbf{t}_s \approx (\mathbf{I} - \mathbf{A})^{-1}\mathbf{B}\mathbf{p_s}, \tag{3.4}$$

$$\mathbf{t_s} \approx \mathbf{R}\mathbf{p_s}, \tag{3.5}$$

where $\mathbf{t_s}$ and $\mathbf{p_s}$ represent the temperature and the power at the steady-state, and the $\mathbf{R}$ matrix represents the *steady-state thermal transfer matrix*. The previous equations help in defining the thermal transfer matrix $\mathbf{R}$ as:

$$\mathbf{R} = (\mathbf{I} - \mathbf{A})^{-1}\mathbf{B}, \tag{3.6}$$

based on the physical relationship between power and temperature on a multi-unit configuration, the $\mathbf{R}$ matrix should be a symmetric matrix with maximum values on the diagonal. Thus, the initialization shown in line 2 of Algorithm 1. The initial guess about the $\mathbf{R}$ matrix is then used to initialize the $\mathbf{B}$ matrix using:

$$\mathbf{B} = (\mathbf{I} - \mathbf{A})\mathbf{R}, \tag{3.7}$$

after initializing the $\mathbf{B}$ matrix, in line 3 to 5 of Algorithm 1 we determine the B matrix by alternating n times between two steps, n being a hyperparameter:

- $\mathbf{P}$-*step*: estimates $\mathbf{p}$(k), the power consumption per-SoC unit, given an initial guess of the $\mathbf{B}$ matrix. This is achieved by solving the quadratic programming optimization shown in line 4 of Algorithm 1. The optimization is solved under two constraints. The first constraint is the positivity constraint of $\mathbf{p}$(k), since it represents power values. The second constraint ensures that the sum of the power consumption of the SoC units at time k, is equal to $\mathbf{p_{tot}}(\mathbf{k})$, the measured total power at time k. The number of unknowns in this step is equal to the number of the target SoC units times the number of timestamps.

- $\mathbf{B}$-*step*: estimates the B matrix using least squares minimization, as shown in line 5 of Algorithm 1, given $\mathbf{p}$(k) from the $\mathbf{P}$-*step* and the $\mathbf{A}$ matrix from line 1 of

Algorithm 1. The number of unknowns in this step is equal to number of elements of the $\mathbf{B}$ matrix, which is the square of the number of SoC units, for which the power is identified.

Simulation and experiment data has shown that a higher number of iterations n brings a better accuracy, however, 10 iterations could be sufficient in most cases.

After identifying the modeling matrices $\mathbf{A}$ and $\mathbf{B}$, in the *Runtime Estimation* step any thermal and power data can be given as input to estimate the power per-SoC unit. The *Runtime Estimation* step estimates the power per-SoC unit by solving the same quadratic programming as in the $\mathbf{P}$-*step*. The user will have to only provide the temperature values per-SoC unit and the total power consumption, this data will be used along with the $\mathbf{A}$ and $\mathbf{B}$ matrices determined in the off-line training step, to determine the power consumption per-SoC unit.

3.2.2 The Alternating-BPI tool

The goal of the Alternating-BPI tool is to make the fine-grain power analysis under various devices seamless and straightforward. This would enable the research community with a tool that helps in providing useful insights, in order to improve modern SoCs. The Alternating-BPI tool [16] , shown in Figure 3.1, puts under the same package Algorithm 1 and the necessary data processing techniques to automate the whole process of the blind power identification.

As shown in Figure 3.1, the tool is composed of an *Off-line Training* step and *Runtime Estimation* step. During the *Off-line Training* the tool processes the data and estimates the $\mathbf{A}$ and $\mathbf{B}$ matrices. Then, during the *Runtime Estimation*, it estimates the power per

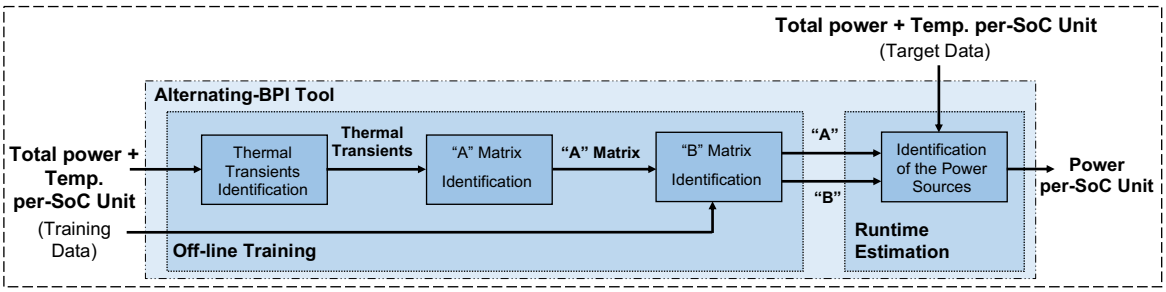

Figure 3.1: The Alternating-BPI tool.

SoC unit of any given data. The composing elements of the tool shown in Figure 3.1 are explained below:

- The training data: is a matrix that includes the total power consumption of the SoC on the first column, and the temperature per-SoC unit on the remaining columns. The training data should be generated by stressing the different SoC units in different patterns, separated by idle states, where a pattern is identified as a combination of active and idle SoC units. The best results are obtained when the data is collected for all the possible patterns, which requires going through all the possible configurations of active and idle SoC units. Collecting the thermal data for different patterns helps in estimating the contribution of each SoC unit to the total power, while the idle states help in creating transient thermal state data, that is used to estimate the $\mathbf{A}$ matrix.

- The target data: this data does not have to follow any specific patterns as required for the training data. The target data represents the data points for which the user wants to identify the power consumption per-SoC unit. It consists of the total power consumption and the temperature per-SoC unit.

- Thermal transients identification: the goal of this step is to identify the thermal transients on the training data. The thermal transients correspond to the natural response temperature variations that are described by Equation 3.2. The thermal

transients are identified by tracking the thermal variation over a sliding window, if the thermal variation exceeds a certain predefined threshold, it is considered as a thermal transient. When a thermal transient is detected, the thermal data points within a predefined range are considered as part of the thermal transient trace and are saved in a matrix. This process is applied to all the training data until all the transient states are detected and recorded. The output of this step, is the thermal transient states.

- $\mathbf{A}$ Matrix identification: This step uses the thermal transients identified in the previous step to identify the $\mathbf{A}$ matrix, as shown on line 1 of Algorithm 1.

- $\mathbf{B}$ Matrix identification: This step estimates iteratively the $\mathbf{B}$ Matrix using the training data given as input, by alternating between the estimation of the $\mathbf{B}$ matrix and $\mathbf{p}$(k), as shown on line 2 to 5 of Algorithm 1.

- Identification of the power sources: This step is about the identification of the power consumption per-SoC unit of the target data, given the $\mathbf{A}$ and $\mathbf{B}$ matrices. This corresponds to the runtime estimation step explained in Algorithm 1. This step is fast enough to generate the power values in real-time, for instance, using an Intel I5-7360U CPU processor, it takes as low as 1.5 ms to generate the power data of 8 SoC units for one timestamp. This makes the proposed technique fast enough to be deployed to generate power predictions in real-time.

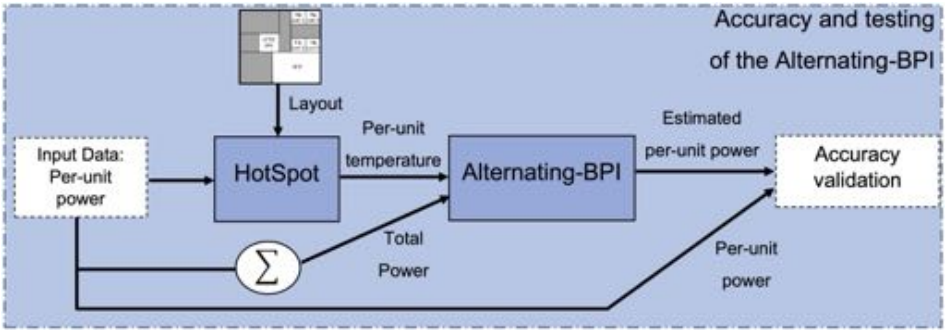

Figure 3.2: The verification and testing flow of the Alternating-BPI.

3.3 Experiments and Results

3.3.1 Experimental setup

Simulation setup: The accuracy of the proposed technique is verified using the HotSpot thermal simulator [47]. As shown on Figure 3.2, for a given design layout, the HotSpot thermal simulator [47] takes as input the per-unit power traces, and produces the corresponding per-unit temperature traces. Figure 3.2 shows that the per-unit temperature traces, from the HotSpot simulator [47], are then taken as input by the proposed Alternating-BPI tool along with the total power. The estimated per-unit power, by the Alternating-BPI tool, is then computed. Finally, the accuracy of Alternating-BPI is computed by comparing the difference between the per-unit power traces given to HotSpot [47] as input, and the estimated per-unit power by the Alternating-BPI.

We follow the same methodology as in [67], and we choose three different floorplan benchmarks:

- 2x2 mesh : a floorplan composed of 4 units with a total maximum power budget of 80 W, and 1 cm x 1 cm as the floorplan dimensions.

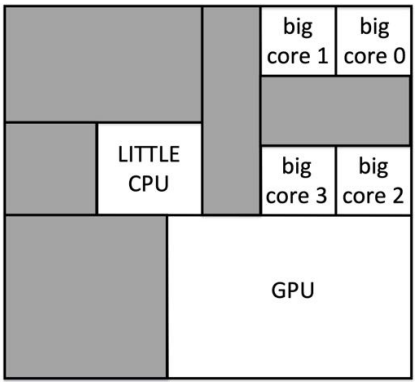

Figure 3.3: Layout of the big.LITTLE+GPU SoC [43] used for the testing of the Alternating-BPI.

- 3x3 mesh : a floorplan composed of 3 units with a total maximum power budget of 80 W, and 1 cm x 1 cm as the floorplan dimensions. This floorplan was chosen to test how the increase in the number of units could affect the per-unit power accuracy.

- big.LITTLE+GPU : as shown on Figure 3.3, this floorplan is composed of 6 units, with a total maximum power budget of 15 W, and 1 cm x 1 cm as the floorplan dimensions. This floorplan was chosen to test how the heterogeneity of the architecture could affect the per-unit power accuracy. Additionally, this floorplan benchmark reflects better the existing SoCs, which are mainly based on heterogeneous architectures.

Development board setup: There is no way to verify the accuracy of the per-unit power estimation on a real device, due to the lack of per-unit power sensors. Actually, the true motivation behind the proposed Alternating-BPI technique is to provide per-unit power estimations, based on the per-unit temperature measurements, due to the non-existence of per-unit power sensors. Despite this, a validation test could still be performed on a real device, by running different workloads and contrasting the per-unit power numbers based on the hardware specification of each unit, which could validate the results. For instance, the power consumption of the GPU should be significantly higher when running

a GPU benchmark, as compared to running a CPU Benchmark. Additionally, we know from the hardware specification that certain CPU cores are designed for power efficiency, while other cores are designed to provide maximum performance, based on this we would expect the power numbers of the power efficient cores to be significantly lower, after taking into consideration the hardware utilization numbers. However, the exact accuracy of the proposed technique would still be verified based on the simulation data.

Thus, the applicability of the proposed Alternating-BPI is tested on the Snapdragon-865 hardware development board [18], shown on the left side of Figure 3.4, which runs using the state of the art Snapdragon-865 SoC. It has a 4+3+1 CPU based on the ARM big.LITTLE architecture. More precisely, the CPU is composed of four "LITTLE" cores that are designed for energy efficient computing, they have a maximum frequency of 1.8 GHz, four "big" cores that are designed to provide maximum performance, three of them run at a maximum frequency of 2.42 GHz, and one big core that runs at a maximum frequency of 2.84 GHz [14, 65]. The SoC integrates: the Adreno 650 GPU, the Qualcomm Hexagon 698 DSP, which is referred to as CDSP in this chapter, and the Qualcomm Spectra 480 image signal processor, which is referred to as SDSP. We divide the previously mentioned hardware blocks to six clusters, as shown in Table 3.2, for which we try to identify the power. In order to get insights that reflect the real behavior of mobile devices, the frequency is dynamically scaled by the default governor of the device during the experiments related to the power analysis .

The thermal traces are collected by reading the SoC embedded thermal sensors using a C code that runs on the device. As shown on the right side of Figure 3.4, we used the Monsoon HV power monitor AAA10F to measure the total power.

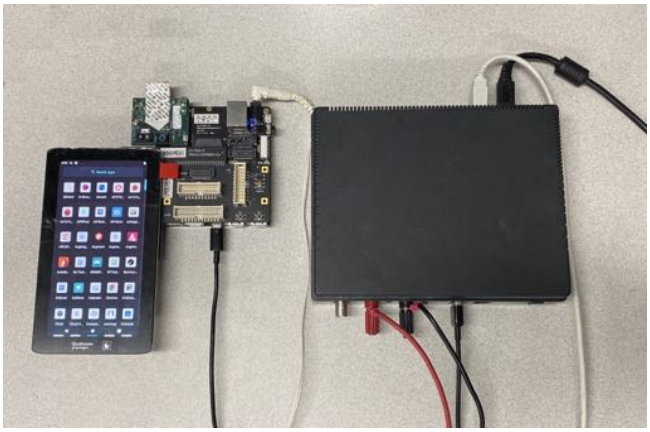

Figure 3.4: The used setup for the experimental verification of Alternating-BPI: the 865-HDK on the left side, and the Monsoon power monitor on the right side.

Table 3.1: The power estimation error of the Alternating-BPI against BPI and BPISS using three floorplan benchmarks.

	BPI [66]	BPISS [72]	Alternating-BPI
2x2 mesh (4 units)	4.42 %	4.40 %	2.44 %
3x3 mesh (9 units)	9.92 %	6.5 %	2.48 %
big.LITTLE+GPU (6 units)	11.19 %	8.84 %	1.92 %

3.3.2 Results

Verification using simulation: We choose three different floorplans as benchmarks, including a big.LITTLE+GPU floorplan, shown on Figure 3.3, which is similar to the architecture of the Snapdragon-865 SoC used later for the experimental validation of the Alternating-BPI.

We compare against BPI [66], which is the first work to introduce a blind approach for the identification of power sources. BPI [66] relies on the non negative matrix factorization (NMF) [50] to identify the $\mathbf{B}$ matrix. However, the accuracy of the NMF [50] output is sensitive to the initialization step. Thus, we additionally compare against another version of BPI [72] that improved the initialization step by relying on the steady state temperatures, which we refer to as BPISS [72].

As shown in Table 3.1, the proposed technique was able to estimate the power with a better accuracy than BPI [66] and BPISS [72] for the three floorplan benchmarks. The power identification is supposed to be more difficult for a higher number of clusters, as well as, for heterogeneous systems like the big.LITTLE+GPU architecture.

As shown in Table 3.1, contrasting the results of the 2x2 mesh with the 3x3 mesh, we realize that as the number of units increases, the estimation error of both BPI [66] and BPISS [72] increased to 9.92% and 6.5%, respectively. On the other hand, the proposed Alternating-BPI was able to estimate the power with a better accuracy, without any significant increase in the estimation error, as the number of units increased. More precisely, the estimation error of the Alternating-BPI for the 2x2 mesh and the 3x3 mesh was 2.44% and 2.48%, respectively.

Contrasting the results of the 3x3 mesh to the big.LITTLE+GPU, we notice that as we moved to a heterogeneous design,the estimation error of both BPI [66] and BPISS [72] increased to 11.19% and 8.84%, respectively. On the other hand, the proposed Alternating-BPI was able to estimate the power with a better accuracy, as low as 1.92%, without any increase due to the heterogeneity of the architecture.

Figure 3.5 shows the predicted power by Alternating-BPI and BPISS [72], as compared to the actual power per-cluster for the big.LITTLE+GPU floorplan benchmark. As shown, BPISS [72] ability to correctly estimate the power varies across the different power pulses and clusters. On the other hand, Alternating-BPI was able to predict with a high accuracy the power of the different clusters, and maintain such level of accuracy across the different power pulses and clusters. For instance, BPISS [72] was able to achieve a much better accuracy for Cluster 3, as compared to the estimation accuracy for Cluster 1 and Cluster 2. On the other hand, the power estimation of the proposed Alternating-BPI for the three clusters was equally accurate, while out-performing the accuracy of the other

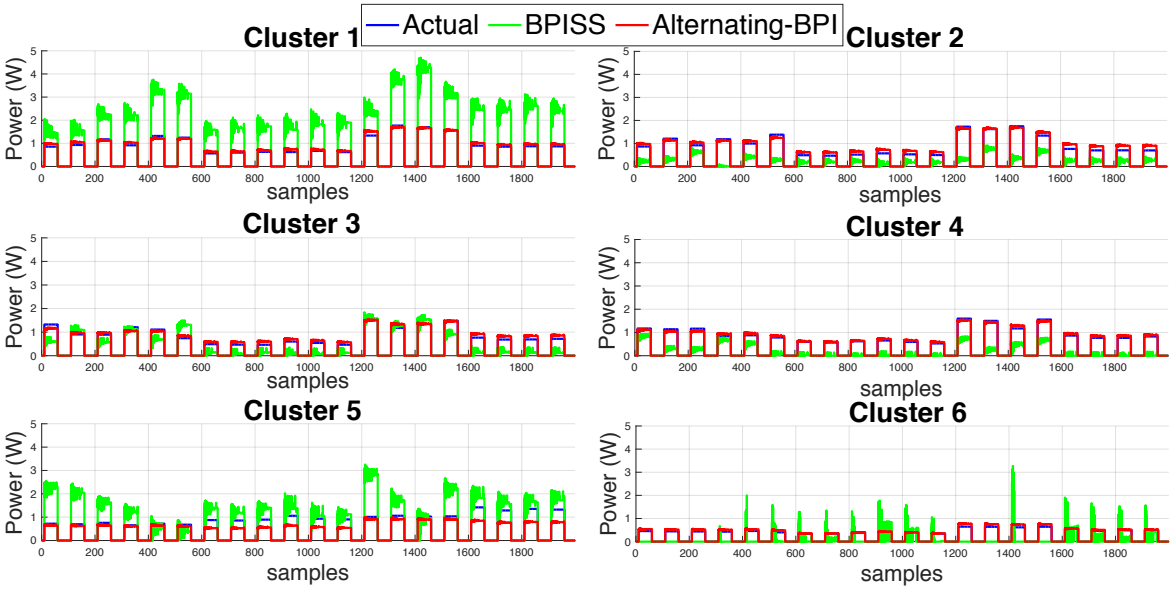

Figure 3.5: The accuracy of BPISS vs Alternating-BPI in predicting the power per-cluster for the big.LITTLE+GPU floorplan.

technique significantly.

The accuracy improvement of the per-unit power estimation of the Alternating-BPI, as compared to BPI [66] and BPISS [72], is mainly due to the better estimation of the $\mathbf{B}$ matrix. Both BPI [66] and BPISS [72] rely on the steady state-thermal data to estimate the $\mathbf{B}$ matrix. On the other hand, the Alternating-BPI relies on the iterative process introduced in Algorithm 1, that uses the whole data to improve the accuracy across the different iterations.

Development board testing: We used the Snapdragon-865 HDK to test the proposed technique by identifying the state space model matrices and estimating the power per cluster. As previously mentioned, we divide the SoC to 6 clusters. As shown on Table 3.2:

- Cluster 1: ”LIT” is composed of the four little cores of the CPU. The four cores run at the same frequency and they could reach a maximum frequency of 1.80 GHz.
- Cluster 2: ”Big” is composed of the first three cores of the big CPU cluster. The

Table 3.2: The hardware blocks composition of each cluster.

	Clus. 1: LIT	Clus. 2: Big	Clus. 3: Big4	Clus. 4: GPU	Clus. 5: CDSP	Clus. 6: SDSP
Hardware Blocks	Core 1 Lit. Core 2 Lit. Core 3 Lit. Core 4 Lit.	Core 1 Big Core 2 Big Core 3 Big	Core 4 Big	GPU	CDSP	SDSP

three cores run at the same frequency and they could reach a maximum frequency of 2.42 GHz.

- Cluster 3: ”Big4” is composed of the fourth core of the big CPU cluster. This core runs at a different frequency than the ”Big” cluster, and could reach a maximum frequency of 2.84 GHz.
- Cluster 4: ”GPU” is composed of the Adreno 650 GPU and could reach maximum frequency of 587 MHz.
- Cluster 5: ”CDSP” is composed of the Qualcomm Hexagon 698 DSP, which is supposed to be the neural engine of the 865 SoC.
- Cluster 6: ”SDSP” the Qualcomm Spectra 480 image signal processor.

The training data for the Snapdragon-865 HDK was generated by coding software kernels that stress the different SoC units in various patterns, while collecting thermal and power data, around 80 pulses were collected. The training data was used to train BPI [66], as well as the proposed Alternating-BPI technique. Even though in practice we can not verify the accuracy of the results, analysing the per-SoC unit power traces as compared to the stress patterns helps in making a guess about the validity of the results. The BPI [66] results for 80 pulses seemed to be invalid, because of the existence of various red-flags. For instance, for the same hardware utilization, the power consumption of the little cluster was predicted by BPI [66] to be higher than the power of the Big cluster. This implies that

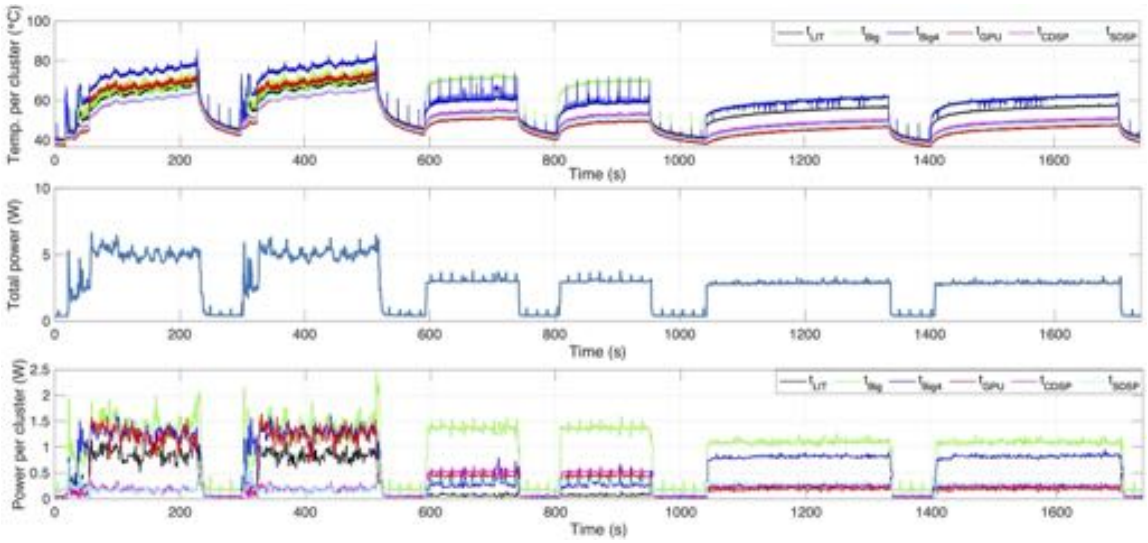

Figure 3.6: The Alternating-BPI estimated power per-SoC Unit of the Snapdragon-865.

BPI [66] needs much more data to start converging towards more reasonable results.

On the other hand, Figure 3.6 shows the temperature per cluster, the total power and the estimated power per-cluster, as estimated by the proposed Alternating-BPI, using 80 pulses, knowing that more data would help in making better power estimation. It has to be mentioned that a hardware SoC unit could highly affect the temperature of other units, based on their location in the layout, and the amount of power being dissipated. In the following we show an analysis of Figure 3.6:

- **[0 s - 500 s]:** we stress the Little, the Big, the Big4 and the GPU clusters, and we can see their estimated power profile increase accordingly. The Big cluster has the highest power profile, which makes sense, because it is composed of 3 big ARM cores. On the other hand, the little cluster shows a low power profile as compared to the big cores and the GPU, which is due to the fact that the little cores are designed to be power efficient and they run at a lower frequency.
- **[500 s - 1000 s]:** we stress only the Big cluster, and as shown in Figure 3.6, Alternating-BPI predicted the Big cluster power to increase to the same level as

Table 3.3: The clusters used by the set of benchmarking apps.

	Clus. 1: LIT	Clus. 2: Big	Clus. 3: Big4	Clus. 4: GPU	Clus. 5: CDSP	Clus. 6: SDSP
Geekbench [8]	Yes	Yes	Yes	No	No	Yes
3DMark [1]	Yes	Yes	No	Yes	No	No
VRMark [21]	Yes	Yes	No	Yes	No	No
AI Benchmark [2]	Yes	Yes	Yes	No	Yes	Yes
AR Civilisations [6]	Yes	Yes	Yes	Yes	No	Yes

in the first 500 seconds. The minor power increase for the other clusters might be related to leakage power and some minor computation triggered by the OS.

- **[1000 s - 1700 s]:** we stress the Big and the Big 4 clusters, and as shown in Figure 3.6, Alternating-BPI was able to predict that the highest power dissipation is coming from the stressed clusters.

Per-unit power characterization of mobile apps: we perform a fine-grain power characterization of various mobile Apps on the Snapdragon-865 SoC using Alternating-BPI. The used benchmarking apps and the clusters they use are shown on Table 3.3. The chosen list includes : CPU, GPU, Virtual Reality (VR), Artificial Intelligence (AI) and Augmented Reality (AR) Apps. These benchmarks stress all the clusters in different patterns and different utilization levels. The bar graph in Figure 3.6 shows the average power per cluster, per benchmarking app:

- **Geekbench [8]:** This benchmark includes 25 multi-threaded workloads of four different sections: cryptography, integer, floating point and memory workloads. As shown in Figure 3.7, the Big and Big4 clusters consume up to 5 Watts, as compared to less than 3 watts for most of the other benchmarking apps. Geekbench relies as well on the little cluster and the SDSP with a combined power consumption of 2 watts. The results show that the little cluster, which is designed for low-power computing, consumes 5x less power than the Big and Big4 clusters. Additionally, the

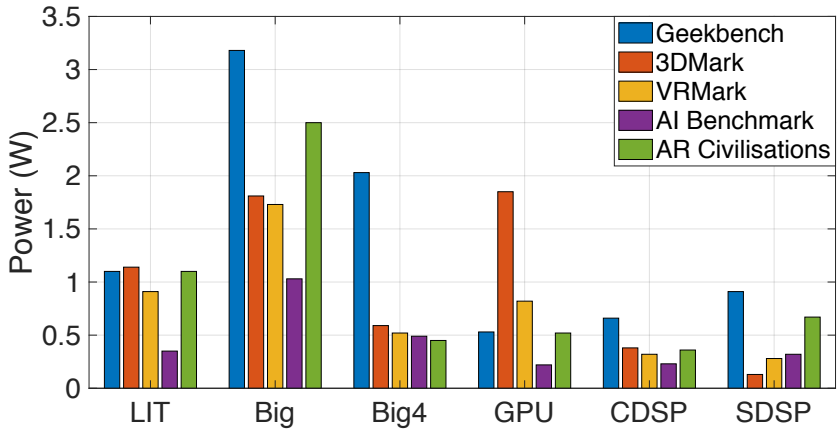

Figure 3.7: The Alternating-BPI estimated power per-SoC Unit of the Snapdragon-865 for the benchmarking apps.

Big4 cluster consumes up to 2x more power than the four cores of the little cluster, which is due to the power-hungry architecture and high operational frequency, that can reach 2.84 GHz. Figure 3.8 shows that the power consumption of the CPU (Little + Big + Big4 clusters) account for more than 75% of the power consumption of the Snapdragon-865 SoC.

- **3DMark [1]:** 3DMark is a GPU-CPU intensive benchmark that tries to mimic gaming apps. As shown in Figure 3.7, the main power goes to the GPU and the Big clusters, with a combined power of 3.6 Watts, representing more than 65% of the total power consumption. Additionally, even when the GPU is highly utilized, its power consumption is 2.5x less than the power consumption of the Big + Big4 clusters. Thus, the CPU Big cluster remains the biggest source of power consumption in the SoC.

- **VRMark [21]:** VRMark is a virtual reality benchmark that is mainly CPU intensive. Figure 3.7 shows that VRMark [21] has the same power profile as 3DMark [1], except for the GPU that consumes 2x less power the VR benchmark, as compared to the GPU benchmark. Figure 3.8 shows that for VRMark more than 65% of the power consumption is coming from the CPU.

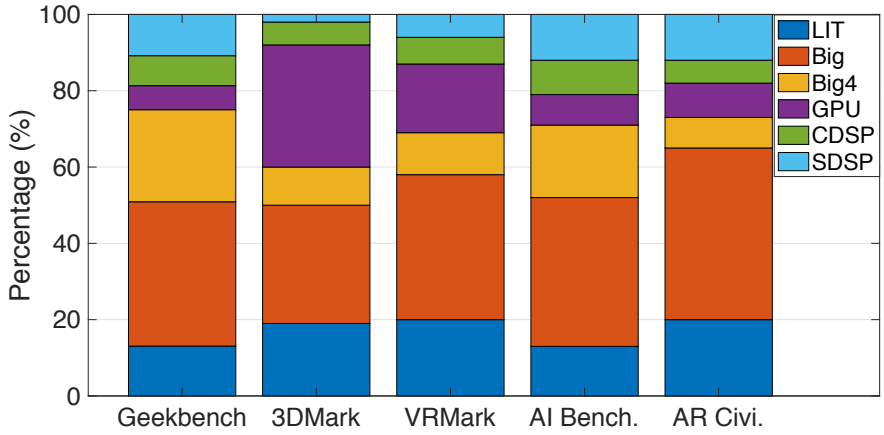

Figure 3.8: The Alternating-BPI estimated percentage power consumption of per-SoC Unit of the Snapdragon-865 for the benchmarking apps.

- **AI Benchmark [2]:** AI Benchmark runs 46 AI computer vision tests, that mainly run on the CPU. Figure 3.7 shows that AI Benchmark [2] has lowest power profile amongst the benchmarking Apps, with 3x less power consumption than Geekbench on the CPU. Figure 3.8 shows that 70% of the power consumption is coming from the CPU.

- **AR Civilisations [6]:** AR Civilisations is an Augmented Reality (AR) educational app. Figure 3.7 shows that the AR App is CPU-hungry, with up to 3 watts consumed by the Big + Big4 clusters, which represents 50% of the total power consumption. Even if it is an AR App, Figure 3.8 shows the GPU represents only 10% of the total power consumption, while the little cluster, which is designed for low-power computing, consumes up to 20% of the total power consumption. This should be related the low 3D rendering utilization of AR Apps, as compared to Gaming Apps.

The main insights that Figures 3.7 and 3.8 show are:

- **The CPU remains the main source of power consumption**: Even with the new computing units that have been integrated to modern Mobile SoCs, like the GPU, the image signal processor and neural engine, the CPU is still the main source of

power consumption, representing around 60% to 75% of the total power for CPU, GPU, VR, AI or AR Apps.

- **The little cluster plays a major role in saving power**: the four cores of the little cluster are highly utilized by most Apps, yet the power consumption of the little cluster is 5x less than the Big and Big4 clusters. The inclusion of more than four cores, based on the LITTLE cores architecture, would strongly enable power efficient computing.

- **The CPU frequency boost has a very low power efficiency**: the 20% frequency boost of the Big4 core, as compared to the cores of the Big cluster, increases the power consumption of the Big4 core by almost 2x. The latter makes sense because the dynamic power is proportional to the voltage square multiplied by the frequency, knowing that the frequency and the voltage on Mobile SoCs are scaled dependently of one another, and this dependence is approximately linear. Thus, the dynamic power is proportional to the cubic of the frequency [48]. This makes the frequency boost highly costly from a power consumption perspective, and makes the power efficiency drop drastically, because a 20% frequency boost will not translate to 2x performance boost.

- **The Augmented Reality Apps have a high CPU power profile and a low GPU power profile**: the Augmented Reality (AR) App shows an average CPU power consumption of 4.5 Watts, which accounts for 75% of the total power consumption of the SoC. On the other hand, the GPU which runs the 3D rendering, accounts for only 9% of the total power consumption.

- **The GPU is highly power efficient**: 3DMark [1] is mainly about 3D rendering and the GPU handles most of its computation, however, it is average power consumption is only 1.85 Watts, which represents only 30% of the total power consumption, and

which is 2x less than the total CPU power when running GPU workloads, and 3.5X less than the total CPU power when running CPU workloads.

Most importantly, the previous experiments demonstrate that the proposed tool could be used to conduct experiments and investigations to characterize the power consumption at a fine grain-level, which is an important step towards making, both the hardware and the software, power-efficient.

3.4 Conclusion

Mobile SoCs lack the ability to sense the power at SoC unit level. The existing power identification techniques rely on certain assumptions, and they lack accuracy and practicality, which makes it challenging to get useful insights about the fine-grain power profiles of mobile SoCs, knowing that such insights are critical for the design and improvement of these SoCs. In this work we proposed a new technique for blind identification of power sources. The technique relies on an Alternating-BPI approach, which allows a faster convergence, a better accuracy and practicality than previous blind identification techniques, as it does not require steady thermal states. The proposed approach decreases the estimation error to 1.9%, as compared to 11.2% for BPI [66]. The accuracy of proposed work was verified using simulation and validated using experimental data on a commercial development board. Additionally, the new approach was used to characterize the per-unit power profile of several bencharmaking Apps on a commercial SoC, including : CPU, GPU, Artificial Intelligence (AI) , Virtual Reality (VR), Augmented Reality (AR) Apps. The power characterization provides insights about the main sources of power consumption and the power efficiency of the different hardware units. Some of the insights include: (1) Even with the new computing units integrated to modern Mobile SoCs, the CPU is still

the main source of power consumption, representing around 60% to 75% of the total SoC power. (2) The little CPU cluster plays a major role in saving power, with a power consumption that is 5x less than the big CPU cluster. (3) The GPU power consumption for AR Apps, represents only 9% of the total SoC power consumption, while the CPU represents 75% of the total SoC power consumption.

Chapter 4

Deconvolutional Neural Network Based Power Map Estimation

In this chapter, we propose an approach for full-chip power map estimation based on Deconvolutional Neural Networks (DCNN). The proposed approach relies only on the usage of the few available embedded thermal sensors on the SoC to estimate the full SoC power map. The contributions of this chapter are as follows:

- We propose to solve the power map estimation problem as an image generation problem using Deconvolutional Neural Networks (DCNN). The proposed DCNN takes as input the thermal measurements from the embedded thermal sensors and the total power to estimate the full SoC power map.
- The proposed technique allows to estimate the power map at a finer spatial granularity than the thermal spatial granularity of the existing thermal sensors. More specifically, it allows to estimate the power even at locations where thermal measurements are not physically available.

- The proposed technique is demonstrated using a commercial SoC while running several benchmarks. The predicted power maps show a 97% similarity (2D correlation) with the power maps estimated using the Alternating-BPI.

The rest of the chapter is organized as follows: Section 4.1 motivates the proposed work. Section 4.2 describes the proposed technique and the architecture of the proposed DCNN. Section 4.3 presents the evaluation results of our technique using different benchmarks on a commercial SoC. Section 4.4 concludes the chapter.

4.1 Motivation

The increasing processing capabilities of mobile system on chip (SoC) is raising critical power and thermal challenges. The increase in runtime power and thermal variations compromise the performance and reliability of integrated circuits. In order to mitigate the effects of the increased power, runtime power and thermal management is used to prevent violations while maximizing the performance. However, the efficiency of the power and thermal runtime management techniques depends on accurate and fine-grain power and thermal sensing. We highlight three main motivations for full-chip power map estimation using a Deconvolutional Neural Network:

- **Lack of fine grain power sensing in mobile SoCs:** Power measurement requires the usage of current sensors. However, the current sensors and their support circuitry incur power, die area and design overheads. These overheads lead to a lack of fine-grain power sensing in mobile SoCs.
- **Limited number of thermal sensors:** Modern mobile SoCs are enabled with multiple thermal sensors. However, the increasing runtime thermal problems require

more sensors to capture temperatures at a wider range of locations, although manufacturers would like to reduce costs by using the fewest number of thermal sensors.

- **Relying solely on a limited number of power and thermal sensors is not sufficiently reliable:** Modern SoCs are complex chips that include multiple heterogeneous units, supporting a large variety of workloads. The power and thermal peak locations in these modern SoCs are non-stationary and are very workload dependent [69]. Thus, making it difficult to rely solely on fixed power and temperature sensors.

Thus, it is essential to devise a technique that could estimate the full-chip power map, without incurring the design, area or power overheads of physical power and thermal sensors.

4.2 Deconvolutional Neural Network Based Power Map Estimation

The proposed approach relies on the idea of treating the power map estimation as an image generation problem using a Decovolutional Neural Networks (DCNN), where a DCNN is trained using supervised learning to generate the power map given a set of input features.

The general approach of the proposed technique relies on the estimation of the SoC power maps using a blind power identification technique during an offline step. The estimated power maps are then used to train a DCNN during the offline training. Finally, the trained DCNN is used to directly estimate the full power map during inference.

Figure 4.1 shows the offline training flow of the DCNN based power map estimation.

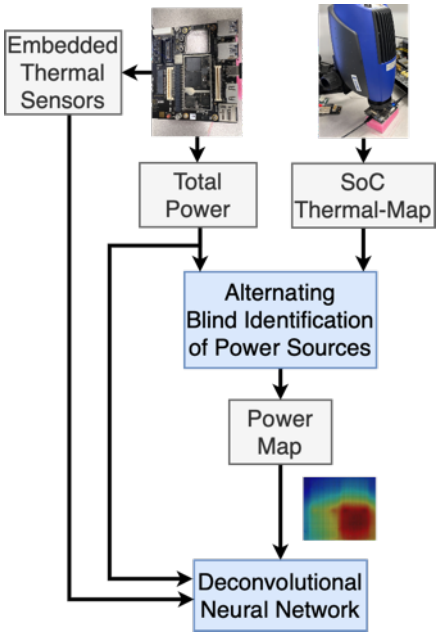

Figure 4.1: The offline training flow of the DCNN based power map estimation.

The flow requires the collection of the following measures:

- SoC thermal-map: The thermal-map of the SoC is collected using an infrared camera. A calibration process is performed to convert the digital level values reported by the camera to actual temperature values, as collected by the internal sensors. The calibration process is achieved by collecting thermal-map data and temperature data using the embedded thermal sensors. Afterwards, the calibration is performed by building a linear regression model that converts the digital levels to temperature values.

- Embedded thermal sensor traces: The internal temperature of the SoC is collected based on the embedded thermal sensors that are available on modern SoCs. We collect the temperature data from 27 thermal sensors distributed across the whole SoC, including temperature sensors on the CPU cores, the GPU and the DSPs. The data from these sensors is collected by running a C-based code in the device, which reads the system nodes that correspond to the target thermal sensors.

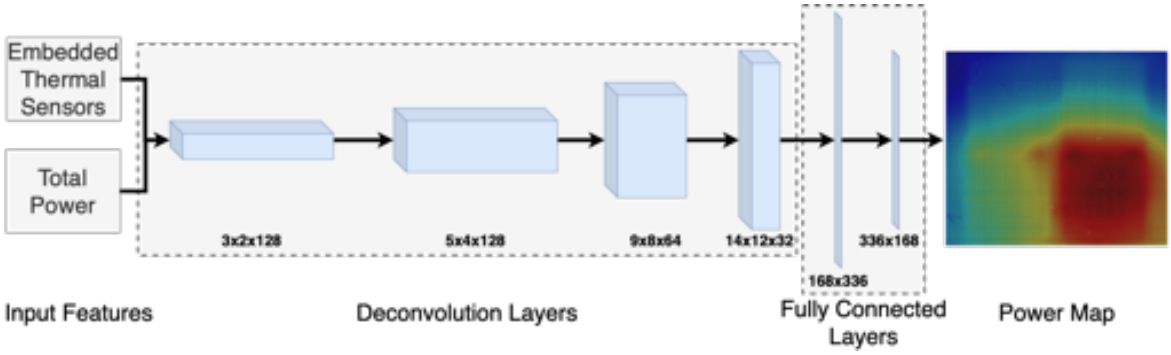

Figure 4.2: The neural network architecture of the proposed DCNN based power map estimation.

- Total power traces: The total power consumption of the SoC could be collected either using the battery current and voltage sensors, or using an external power measurement tool.

The SoC thermal-map, the embedded thermal sensor traces and the total power are collected while running various workloads.

Afterwards, as part of the offline training of the DCNN, we generate the power maps corresponding to the collected thermal maps. The power maps are generated to be used as a target output during the offline training. More precisely, as shown in Figure 4.1, the SoC thermal-map and the total power consumption are given as input to the Alternating-BPI to generate the SoC power map data.

As shown in Chapter 3, the Alternating-BPI can be used to estimate the per-unit power consumption based on the per-unit thermal values, and the total power consumption. In our case, each pixel from the thermal-map given as input to the Alternating-BPI is considered as a power source. Finally, as shown in Figure 4.1, the Deconvolutional Neural Network is trained using the embedded thermal sensors data and the total power as input features, and the power map data generated by the Alternating-BPI is used as a target output.

Table 4.1: DCNN parameters used in this work.

Layer	Kernel	#Output	Activation
Deconvolution Layer	3×2	3×2×128	ReLU
Deconvolution Layer	3×3	5×4×128	ReLU
Deconvolution Layer	5×4	9×8×64	ReLU
Deconvolution Layer	6×5	14×12×32	ReLU
Fully Connected Layer	-	336	ReLU
Fully Connected Layer	-	168	ReLU

Figure 4.2 shows the architecture of the proposed DCNN. The network is composed of four deconvolutional layers and two fully connected layers. The output width and height keep increasing throughout the deconvolutional layers, until it reaches the target output size, which is 14×12 in this case. On the other hand, the channel depth keeps decreasing from 128 channel to 32 channel at the last deconvolutional layer. Then, the output of the deconvolutional layers is given as input to two fully connected layers. The size of the output of the last fully connected layer is 168, which is reshaped to 14×12 power map.

The specific parameters of the proposed DCNN architecture is shown in Table 4.1. The parameters include the layer type, the kernel size, the output size, and the type of activation function being used for the output of that layer. The Mean Square Error (MSE) was the loss function used during the training. The used optimizer was the Adaptive Moment Estimation, usually referred to as the Adam Optimizer, with a learning rate of $2e^{-4}$.

Different parameters for the architecture have been investigated, including different kernel sizes, activation functions, loss functions and optimizer. The hyper-parameter tuning lead to choosing the previously mentioned parameters, which achieved the best performance.

Figure 4.3: The used FLIR SC5000 Series thermal camera setup.

4.3 Experiments and Results

4.3.1 Experimental setup

Figure 4.3 shows the FLIR SC5000 Series thermal camera, which is used to perform the thermal-map data collection. The collected thermal-maps have a sampling rate of 100 Hz and 455×402 resolution, which are then sampled down to 14×12 using Nearest-neighbor interpolation.

We use the Snapdragon-865 hardware development board [19], which runs using the state of the art Snapdragon-865 SoC, used in more than 30 state of the art android mobile devices. It has a 4+3+1 CPU based on the ARM big.LITTLE architecture. The SoC integrates: the Adreno 650 GPU, and the Qualcomm Spectra 480 image signal processor, which is referred to as SDSP. The thermal traces are collected by reading the SoC embedded thermal sensors using a C code that runs on the device. For the purposed of this work,

Table 4.2: The different benchmarks used to evaluate the accuracy of the proposed work.

	Benchmark type
SQLite	An integer workload that executes SQL queries against an in-memory database.
SGEMM	A floating point workload that performs general matrix multiplication.
Face Detection	A floating point workload that performs face detection.
Speech Recognition	A floating point workload that performs recognition of arbitrary English speech using PocketSphinx.
Memory Copy	A memory workload that runs the memcpy() routine using different sizes and randomized offsets.
Memory Bandwidth	A memory workload that measures sustained memory bandwidth.
3DMark	A GPU benchmark that performs 3D rendering and real-time ray tracing.

we collected the internal SoC temperature from 27 thermal embedded sensors distributed across the SoC, including the temperature from the CPU cores, the GPU and the DSPs. We used the Monsoon HV power monitor AAA10F to measure the total power [10].

In order to train the proposed DCNN, we use the Pytorch Framework. Table 4.2 shows the different benchmarks used to evaluate the accuracy of the proposed work. The benchmarks used include integer, floating point, memory and GPU workloads. The training and testing data set is composed of 1600 power maps, which are divided to 80% for the training stage, and 20% for testing.

Table 4.3: The accuracy of the power map estimation of the proposed work using the training and testing data sets.

	Mean Absolute Error (mW)	2-D Correlation Coefficient (%)
Training dataset	12.3	97.4 %
Testing dataset	10.74	97.2 %

4.3.2 Results

Accuracy metrics: In order to evaluate the accuracy of the predicted power maps, we compare the predicted power maps by the DCNN against the power maps estimated using the Alternating-BPI. We use two accuracy metrics:

- The Mean Absolute Error (MAE) = $\frac{\sum_{i=1}^{n}\sum_{j=1}^{m}|x_{ij}-y_{ij}|}{n\times m}$
- The 2-D Correlation Coefficient = $100 * \frac{\sum_{i=1}^{n}\sum_{j=1}^{m}(x_{ij}-\overline{x})(y_{ij}-\overline{y})}{\sqrt{(\sum_{i=1}^{n}\sum_{j=1}^{m}((x_{ij}-\overline{x})^2)(\sum_{i=1}^{n}\sum_{j=1}^{m}((y_{ij}-\overline{y})^2)}}$

The higher the 2-D correlation coefficient the higher the similarity between the two images being compared, with 100% being the correlation coefficient of one image with itself.

Evaluation results: Table 4.3 shows the accuracy of the power map estimation of the proposed work using the training and testing data sets. Table 4.3 shows that the MAE was 12.3 mW for the training data set, with a correlation coefficient as high as 97.4%. In the testing data-set the MAE was 10.74 mW, with a 97.2% correlation coefficient.

In order further evaluate the proposed approach, we conduct a per-benchmark accuracy evaluation. Table 4.4 shows the per-benchmark accuracy of the power map estimation of the proposed work. Table 4.4 shows that the MAE varies between 3.5 mW to 16.9 mW, with most values between 5.4 mW and 13.2 mW. Additionally, Table 4.4 shows that the

Table 4.4: The per-benchmark accuracy of the power map estimation of the proposed work.

	Mean Absolute Error (mW)	2-D Correlation Coefficient (%)
SQLite	8.23	99.4 %
SGEMM	13.2	97.6 %
Face Detection	5.4	99.7 %
Speech Recognition	7.5	99.7 %
Memory Copy	3.5	99.9 %
Memory Bandwidth	8.6	92.1 %
3DMark	16.9	95.1 %

2-D correlation coefficient varies between 92.1% and 99.9%, with most values between 95.1% and 99.7%.

Figure 4.4 and Figure 4.5 show the ground truth power map vs the predicted power map by the proposed work, for the Face Detection and the Speech Recognition benchmarks, respectively. As shown on Figure 4.4 and Figure 4.5, the proposed work was able to predict with a high accuracy the values and the exact location of all the high power density locations. Additionally, Figure 4.4 and Figure 4.5 show that the proposed work has less accuracy for the power estimation of units whose power is less than 100mW.

4.4 Conclusion

In this chapter, we observed that it is essential to devise a technique that could estimate the full-chip power map, without incurring the design, area or power overheads of physical power and thermal sensors. We took a first towards solving this problem by proposing an approach for full-chip power map estimation based on Deconvolutional Neural Networks (DCNN).The proposed approach relies only on the usage of the few available embedded thermal sensors on the SoC and the total power to estimate the full SoC power map. The

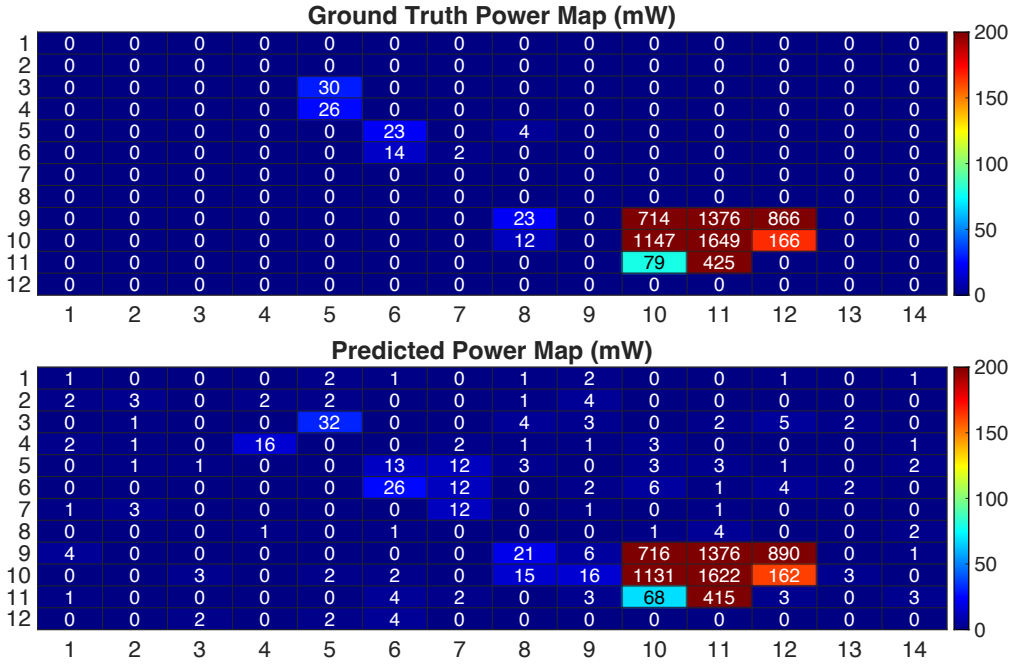

Figure 4.4: The ground truth power map vs the predicted power map by the proposed work for the Face Detection benchmark.

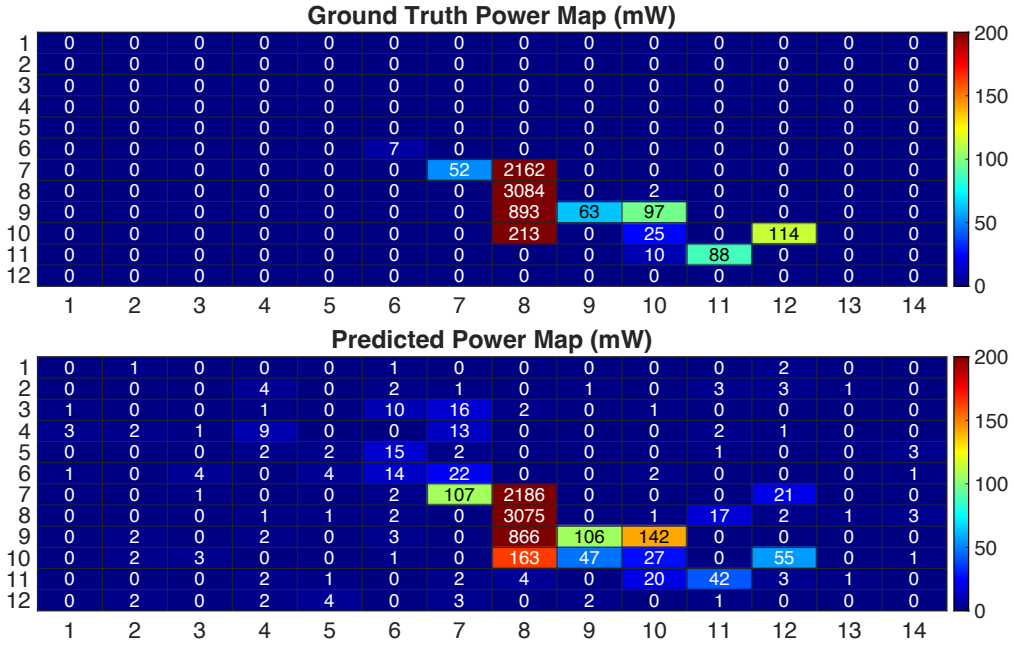

Figure 4.5: The ground truth power map vs the predicted power map by the proposed work for the Speech Recognition benchmark.

proposed technique allows to reconstruct the power map at a finer spatial granularity than the thermal spatial granularity of the existing thermal sensors. More specifically, it allows to estimate the power even at locations where thermal measurements are not available. The proposed technique is demonstrated using a commercial SoC while running several benchmarks. The predicted power maps show a 97% similarity (2D correlation) with the power maps estimated using the Alternating-BPI.

Chapter 5

Power and Hardware Characterization for Augmented Reality Applications

In this chapter we perform a power and hardware characterization of Augmented Reality Apps and we design ARBench, an Augmented Reality benchmark for mobile devices. To summarize, the contributions of this chapter are as follows:

- We characterize the hardware utilization of several AR Apps using a commercial device. Then, we analyze the power consumption per-hardware unit and per-AR process.
- While existing AR benchmarks mainly target Desktop computers, we design and develop ARBench, the first AR benchmark that measures the AR performance of mobile devices. The benchmark incorporates different AR workloads that stress multiple hardware units of the SoC (CPU, GPU, DSP, etc), and measures the individual score for each AR workload.

- We provide insights about the ability of existing mobile devices to run AR workloads, by using ARBench to evaluate the performance of several commercial mobile devices and Android Operating Systems. The performance results are shown for each individual AR workload.

- We use ARBench to perform a phase analysis to identify a set of canonical phases that could be used to model and characterize AR workloads.

- We study the performance and power trade-off of different multi-core CPU configurations, and provide insights about the most efficient multi-core configuration that could run the AR workloads, while meeting the performance requirements.

The rest of the chapter is organized as follows: Section 5.1 motivates the proposed work. Section 5.2 introduces the experimental setup. Section 5.3 introduces the hardware utilization and power characterization of mobile AR Apps. Section 5.4 presents the design and development of ARBench. Section 5.5 validates the results provided by ARBench, and evaluates the AR performance of different mobile SoCs and Android operating systems. Section 5.6 presents the phases analysis of AR workloads. Section 5.7 shows the power and performance trade-offs of different CPU multi-core configurations. Section 6.4 concludes the chapter.

5.1 Motivation

The global Augmented Reality (AR) market is expected to experience a remarkable growth. The AR market worth $7bn in 2020, is anticipated to generated $152bn by 2030. The wide range of applications and use cases that AR offers has made it easily adoptable in various sectors, including manufacturing, education, service, healthcare and supply chain. It is

expected to make a huge impact on our daily lives by improving the human efficiency in performing different tasks, while bringing a more immersive experience.

In 2021, the number of AR active users on mobile has increased to 810 million, because many consumers already own an AR capable smartphone. The release of software development kits for the design of AR Apps, like ARCore and ARkit, for Android and iOS is making mobile devices the ultimate platform to run AR Apps.

Unlike regular mobile Apps, AR Apps rely on the simultaneous usage of different hardware-units and the execution of multiple processes. An AR app needs to consistently read the positional and GPS data, perform camera processing, run mapping and tracking algorithms, 3D rendering of the virtual object. These key differences, make the existing mobile devices less suitable to offer a seamless AR experience.

In fact, the execution of AR Apps on mobile devices is facing some key challenges. The battery lifetime has become one of the top usability concerns of mobile devices, the increase in computing capabilities has outpaced the battery saving development. The battery lifetime issue is exacerbated when using AR Apps because of their simultaneous usage of different hardware-units, in addition to the camera that needs to be always turned on. Furthermore, most AR Apps are about the mixing of a digital object with the real world, after placing the object, the App needs to keep track of it, which is achieved using marker-less tracking. The marker-less tracking is a difficult problem because it requires a complex understanding of the 3D space, thus, making this essential AR feature compute intensive. For instance, the players' report of Pokemon GO, an AR gaming app used by more than 160 million users, has shown rising concerns about the draining of phone battery in the background.

While few challenges are facing the widespread usage of AR Apps, we still lack a

good understanding of several aspects, including : A) the utilization of the different hardware units in the SoC by AR Apps, B) the main hardware and software sources of power consumption while running AR Apps, C) the performance of AR workloads on existing commercial mobile devices, D) the difference between the various AR workloads at the hardware and system level, E) the multi-core CPU configuration that allows to efficiently run the different AR workloads.

5.2 Experimental setup

In order to provide insights that truly reflect the existing platforms, all the chapter experiments are performed on commercial mobile devices. We use the Snapdragon-865 hardware development board [19] to run experiments that require power and hardware utilization collection. In order to evaluate the performance of different mobile devices, we use BrowserStack [7] to remotely access a variety of commercial mobile devices and run ARBench.

The Snapdragon-865 hardware development board [19] runs using the state of the art Snapdragon-865 SoC, used in more than 30 state of the art android mobile devices. It has a 4+3+1 CPU based on the ARM big.LITTLE architecture. More precisely, the CPU is composed of four Little cores designed based on the ARM A55 architecture for energy efficient computing, they have a maximum frequency of 1.8 GHz, four Big cores designed based on the ARM A77 architecture, and they are designed to provide maximum performance, three of them run at a maximum frequency of 2.42 GHz, and one big core that runs at a maximum frequency of 2.84 GHz, which we refer to as the Boost core. The SoC integrates: the Adreno 650 GPU, and the Qualcomm Spectra 480 image signal processor, which is referred to as SDSP. The thermal traces are collected by reading the

SoC embedded thermal sensors using a C code that runs on the device. We used the Monsoon HV power monitor AAA10F to measure the total power [10]. The hardware utilization per-unit is collected using the Snapdragon Profiler [20].

5.3 Hardware utilization and power characterization of mobile AR apps

In this section we characterize the hardware utilization per-SoC unit of several AR Apps using the Snapdragon-865 hardware development board [19]. Then we analyze the power consumption of each hardware unit by applying the blind identification of power sources [72]. Finally, based on the power consumption per-SoC unit and the hardware utilization numbers, we identify the power consumption per AR process.

Characterization of the per-hardware unit utilization: In order to provide an exhaustive characterization of the existing AR Apps, we picked 11 AR Apps from different categories. As shown on Table 5.1, we target educational, e-commerce, gaming and development AR Apps. As described on Table 5.1, the target AR Apps have different use-cases and operate differently. AR Apps usually merge the camera input with some virtual objects, however, the type and the number of the virtual objects integrated in the scene, as well as the way the user interacts with the virtual objects differ from one AR app to another. The set of AR Apps chosen in Table 5.1 takes this into account by including AR Apps in which the user-virtual object interactions are different, and the inserted virtual objects ranges from single and multiple objects, face filters and image filters.

We run the AR Apps from Table 5.1 while collecting the hardware utilization of the different hardware units using the Snapdragon Profiler [20]. Figure 5.1 shows the per-

Table 5.1: The set of Augmented Reality Apps used for the hardware characterization.

AR Apps	Description	Category	Virtual Object
Civilisations AR [6]	Placement of ancient artifacts, the virtual object can be rotated and scaled.	Educational	Single object
Mission to Mars [12]	Placement of 3D rovers and 3D space, the virtual object is animated.	Educational	Multiple objects
Augmently [5]	Placement of 3D furniture items, the virtual object can be rotated and scaled.	E-commerce	Multiple objects
AR 3D Animal [3]	Placement of 3D animals, the virtual object can be rotated.	Educational	Single object
Knightfall AR [11]	Placement of a 3D battlefield, the virtual objects movement can be controlled.	Gaming	Multiple objects
Hello AR [9]	Placement of 3D objects, the virtual objects can not be rotated or scaled.	Development	Multiple objects
Samsung AR [15]	Placement of 3D mobile devices, the virtual object can be rotated.	E-commerce	Single object
YouCam [22]	Projection of commercial makeup products on the user's face.	E-commerce	Face filters
Sketch AR [17]	Projection of a sketch to assist the users in improving their drawing skills	Educational	Image filter
Dragon AR [4]	Projection and interaction with a 3D Dragon, the virtual object is animated and can be rotated.	Gaming	Single object
Monster Park [13]	Placement and the interaction with 3D Dinosaurs, the virtual objects are animated.	Educational	Multiple objects

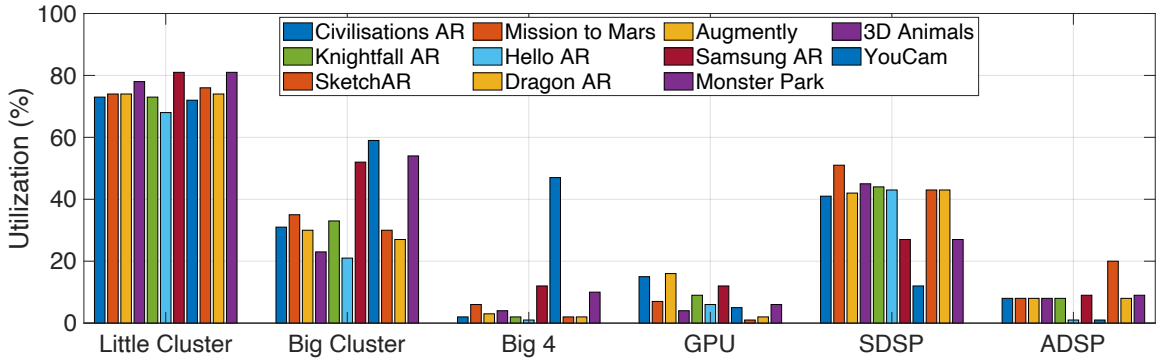

Figure 5.1: Per-Hardware unit utilization of different AR Apps running on the Snapdragon 865 SoC.

hardware unit average utilization for the different tested AR Apps on the Snapdragon 865 SoC. Few insights can be observed from the figure, including:

- There is a high correlation in the per-unit hardware utilization across the different AR Apps: comparing the per-unit utilization of the different AR apps, at the exception of certain cases, we notice that most AR apps have similar utilization numbers. For instance, most of the AR Apps have an average of 75% utilization at the Little cluster. This implies that frameworks like ARCore, which are used in the development of these AR Apps play a major role in determining the computation intensity of the AR apps, regardless of the App use-case.

- The AR Apps mainly rely on the Little cluster, Big cluster and the image processing unit: Figure 5.1 shows that the utilization exceeds 35% on average for the CPU Little and Big cluster and the image processing unit, which we refer to as the SDSP. This implies that AR Apps mainly rely on these clusters to provide the AR experience.

- Even though AR Apps involve the rendering of a 3D virtual object, the GPU is not highly used. However, this does not necessarily reflect the rendering requirements for future AR Apps, which might include the insertion of rendering intensive virtual objects, adding to this the fact that the upcoming commercial devices require a

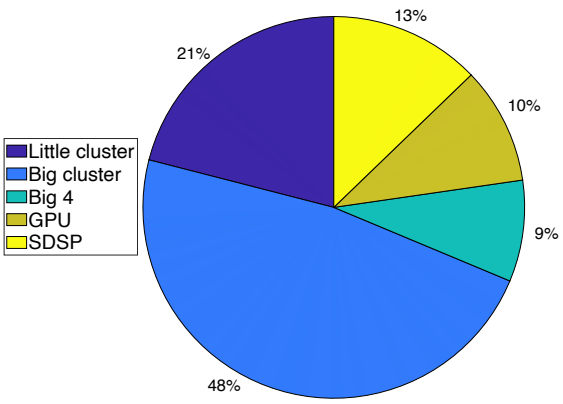

Figure 5.2: Per-Hardware unit power consumption of Civilisations AR on the Snapdragon 865 SoC.

screen refresh rate of 120 FPS, rather than 60 FPS.

Per-hardware unit power consumption: Mobile devices usually lack the ability to measure the power at the hardware unit level. However, they are usually enabled with thermal sensors. Thus, in order to measure the power consumption per-hardware unit we apply the Alternating BPI algorithm from Chapter 3 to identify the power consumption at the hardware-unit level based on the total power consumption and the per-unit thermal measurements.

Figure 5.2 shows the percentage power consumption per-hardware unit, while running Civilisations AR [6] on the Snapdragon 865 SoC. Figure 5.2 shows:

- The CPU is the main source of power consumption: 77% of the total SoC power consumption is dissipated at the CPU level. More precisely, the big cluster consumes 48% of the total power, while the little cluster accounts for 21% of the total power consumption
- The power efficiency of the little cluster is well exploited by AR Apps: even though, the average utilization of the little cluster is around 78%, its power consumption ac-

counts for only 21% of the total SoC power consumption. Actually, the Little cluster cores are designed to be power efficient, and our power evaluation and hardware characterization show that the Little cluster is well exploited by AR Apps.

Per-process power consumption: Based on the hardware utilization of the different processes, and the identified power consumption per-hardware unit using the Alternating BPI algorithm, we can identify the power consumption of the different AR processes. This is achieved by multiplying the process utilization by the power consumption of its corresponding unit, as estimated by the BPI [66].

In order to demonstrate this capability, we collect the hardware utilization of the different Augmented Reality processes, using Snapdragon Profiler, while running AR Civilisation on the 865 HDK. Using the Alternating-BPI we get the power consumption per hardware block, and then we partition the power across the different tasks. The processes in question are:

- Operating system processes: This includes all system related threads. Most of these threads are kernel threads, and they usually run on the little cluster and the Big4 core, taking around 19% of the little cluster utilization, and roughly 1% of the Big4 core utilization.
- Image processing processes: This includes all the threads that handle the camera sensor readings, and the image filtration and calibration. These processes usually run on the little cluster and the SDSP. They take around 54% of the utilization of the little cluster, and 20% of the utilization of the SDSP.
- SLAM + App workload processes: This includes threads are related to Simultaneous localization and mapping (SLAM) algorithm, which includes computer vision and object tracking, positional and GPS data reading, synchronization and sensor fusion.

These threads usually run on the Big cluster, taking around 26% of its utilization, and on the Big4 core, taking 10% of its utilization in average.

- 3D Rendering processes: This includes the processes that run on the GPU to execute the 3D rendering the Augmented Object. This usually takes around 15% of the GPU utilization in the case of AR Civilisation.

- Other: This includes all other threads that are not part of the previously mentioned processes, and that get triggered randomly.

Figure 5.3 shows the per-process power consumption of Civilisations AR [6], while running on the Snapdragon 865 SoC. The main takeaways from Figure 5.3 are as follows:

- Operating system processes: They consume around 5% of the total power, which is mainly due to the fact that the system threads run on the little cluster, making it power efficient to run the system tasks.

- The Simultaneous localization and mapping (SLAM) and the other related App processes are the main source of power consumption: The mapping and tracking of the 3D virtual objects mainly run on the big cluster and account for 54% of the total power consumption.

- The image processing task is the second most important source of power consumption: By running on the little cluster and the SDSP, the image processing processes account for 27% of the total power consumption.

- The 3D rendering, which runs on the GPU, accounts for only 10% of the total power consumption.

These insights could be used to improve the power efficiency of Augmented Reality

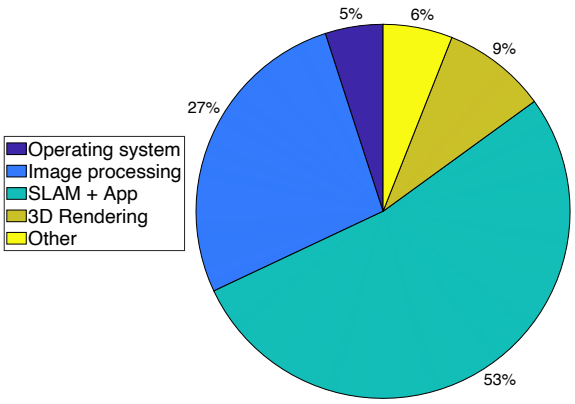

Figure 5.3: Per-AR process power consumption of Civilisations AR on the Snapdragon 865 SoC.

Apps by focusing on the most power hungry tasks, for instance, one direction could be to off-load the SLAM algorithm to the cloud.

5.4 Augmented Reality Benchmark

The study in section 5.3 has shown that AR Apps rely on the simultaneous usage of different hardware units, which makes them different from regular CPU or GPU Apps. The performance and the power consumption are determined by many concurrent processes running on different hardware units, which implies that the measured performance by general-purpose benchmarks might not reflect the true AR user-experience provided by commercial mobile device. Being able to accurately measure the performance of different AR workloads on mobile devices is a crucial step towards the optimization of AR Apps and the improvement of the user-experience.

The AR Apps from Table 5.1 show a multitude of AR use cases from different App categories, and as shown on Figure 5.1 these Apps stress different hardware units simultaneously. Thus, an AR benchmark should include several AR workloads from different AR

use cases, rather than relying on a single scenario. Additionally, the benchmark should be able to stress the different hardware units simultaneously. Furthermore, an AR benchmark should consider ongoing and future progress of AR workloads, for instance, future AR Apps might involve the insertion of rendering intensive virtual objects with target refresh rate of 120 FPS, which implies that the benchmark should include workloads with higher rendering requirements than existing AR Apps.

While taking into consideration the previously mentioned requirements, we design and develop ARBench, the first dedicated Augmented Reality Benchmark for mobile devices and we validate its results. The proposed benchmark includes various AR workloads, such that each AR workload evaluates a particular aspect of Augmented Reality, while simultaneously stressing multiple hardware units of the SoC.

ARBench measures and reports the performance of each particular AR workload by simulating an actual user session in different categories of AR applications. The benchmark suit consists of six benchmarks, each of which evaluates the performance of one of four applications. The applications used are open-source samples provided with Google ARCore, and includes HelloAR, Augmented Faces, Augmented Image and Object Recognition.

The development process was composed of two stages:

- **Benchmark workload generation:** in order to stress mobile devices with AR workloads, the data associated with the workload needs to be generated. The benchmark workload data includes recordings of the user input, such as the user screen touches coordinates, and includes recordings of camera's video stream and IMU data. This is achieved by using the recording API offered by ARCore. The benchmark workload data was generated by adding the record feature to 6 different AR Apps, and

Table 5.2: The description and the objective of each benchmark of ARBench.

Benchmark	Description	Duration	Objective
Object Generation	Inserts a single virtual object on a surface and views the object from a close distance, while moving the mobile device capture angle.	32s	Tests basic functionality with low performance requirements on AR workloads that involve the insertion and rendering of single 3D virtual object.
Multi-Object Tracking	Maps out a large surface and inserts multiple objects in a widespread area. Spends most of the time moving around the space and viewing objects from different distances and angles.	162s	Evaluates performance on AR workloads with intensive tracking and graphics, that involve the insertion and tracking of several 3D virtual objects.
Scene Overloading	Keeps inserting a very large number of objects in a densely packed fashion so that many objects are onscreen at the same time.	112s	Evaluates performance on AR workloads where very intensive graphic workloads are the bottleneck for performance.
Augmented Faces	Applies a face filter to a human face visible in the frame. The human face keeps moving and changing capture angles throughout the benchmark.	27s	Evaluates performance of face detection including identifying the different human face features.
Augmented Image	Inserts a virtual photo frame when viewing a specified target image. The virtual photo is viewed from different angles and distances.	99s	Evaluates performance of 2D image detection and tracking.
Object Recognition	Scans real objects in a room and applies labels to identify them. The device keeps moving throughout the experiment, such that it captures new objects.	76s	Evaluates performance on AR workloads that involve heavy computation for object detection and recognition.

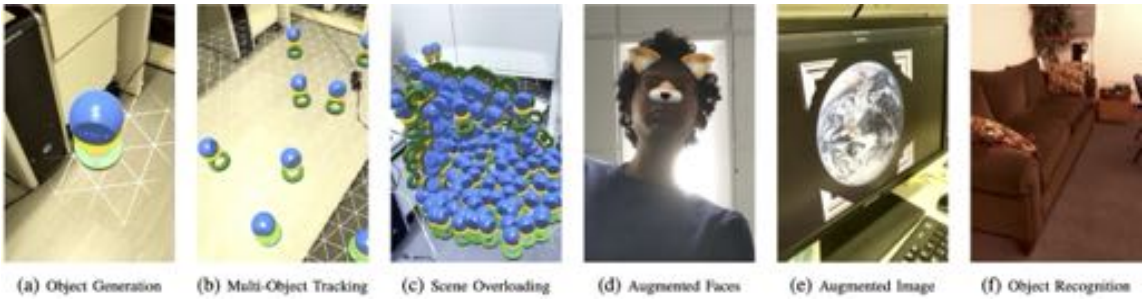

Figure 5.4: Outputs of the six benchmarks of the ARBench.

running each AR App with a particular pattern of user interaction, while recording the previously mentioned data.

- **ARBench development:** after generating the AR workload data, ARBench was designed by combining the 6 different Apps, and adding the playback features of ARCore while collecting the performance metrics. More precisely, ARBench was designed such that the playback functions, offered by ARCore, take as input the recordings generated in the previous step and reconstruct the user behavior, while displaying the recorded data. Additionally, we included the computation of the frame rate for each benchmark, which is used as the performance metric.

In summary, for each benchmark of ARBench, the corresponding application launches with the recording for that benchmark. The application recreates the prerecorded session, performing all the necessary tracking and rendering. The performance metrics for each benchmark are reported once the benchmark has finished running.

The 6 benchmarks of ARBench are summarized in Table 5.2. Figure 5.4 illustrates the expected output of each benchmark. In the following, we introduce the different ARBench benchmarks:

(a) Object Generation: In this benchmark, the camera is pointed at a flat surface and a single object is inserted on the surface, as shown on Figure 5.4. The object is

inspected by moving the camera towards and away from it. This benchmark evaluates the performance of basic AR functionality that includes SLAM and marker-less tracking, with the insertion and rendering of single 3D virtual object.

(b) Multi-Object Tracking: First, a large surface is mapped out by moving the camera around the room. Then as shown on Figure 5.4, multiple objects are inserted randomly over the surface. The camera then moves around the room, keeping track and viewing the objects from some distance and at different angles. This is meant to represent a usage pattern of an AR application, where the user's interaction with the app can be separated into three phases: (1) mapping out the environment, (2) adding virtual objects, (3) interacting with the virtual objects. This benchmark involves the insertion and tracking of several 3D virtual objects.

(c) Scene Overloading: As shown on Figure 5.4, multiple virtual objects are repeatedly inserted in a small region of the room. In total over 200 objects have been added the scene. This benchmark evaluates performance on AR workloads where very intensive graphic workloads are the bottleneck for performance.

(d) Augmented Faces: In this benchmark, a face filter is applied to the face visible in the image, as shown on Figure 5.4. The application performs continuous face tracking and detection, including locating different features like the nose and forehead and performs some light rendering work.

(e) Augmented Image: Searches for a specific 2D target image in the environment and augments it with a virtual photo frame, as shown on Figure 5.4. This benchmark evaluates performance of face tracking and identification of the different human face features.

(f) Object Recognition: The camera scans real objects in a room and applies labels to identify them, as shown on Figure 5.4. This benchmark evaluates performance on

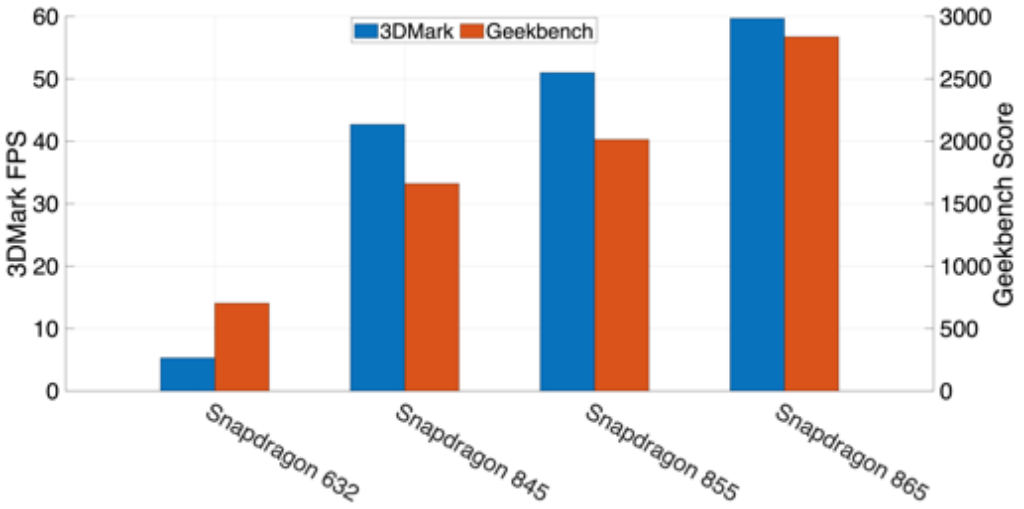

Figure 5.5: 3DMark and Geekbench performance numbers on different Snapdragon SoCs.

AR workloads that involve heavy computation for object detection and recognition.

5.5 AR Benchmarking of Commercial Mobile Devices

In this section we validate the performance results provided by ARBench, by comparing against the scores of CPU and GPU mobile device benchmarks. Afterwards, ARBench is used to evaluate the AR performance of different commercial mobile SoCs and Android operating systems.

Validation of ARBench: In order to validate ARBench, we run it on four different commercial SoCs and we compare the reported performance results to the scores reported by 3DMark and Geekbench, which are CPU and GPU benchmarks, respectively. The per-benchmark performance results reported by ARBench should not be similar to the 3DMark and Geekbench scores, because ARBench evaluates the simultaneous usage of different hardware units, as it occurs on AR workloads. However, the results across the different SoCs need to be correlated to reflect the CPU and GPU performance improvements.

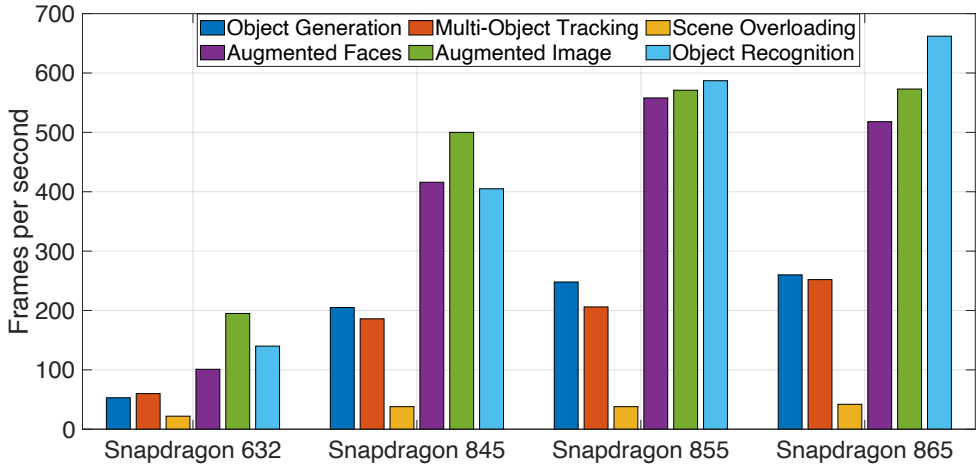

Figure 5.6: The per-benchmark AR performance of different Snapdragon SoCs while running the proposed benchmark.

Figure 5.5 shows the 3DMark and Geekbench performance number on different Snapdragon SoCs. The Snapdragon SoCs, from left to right on the x-axis, are ordered from the oldest to newest generation, such that the Snapdragon 865 is the state-of-the-art SoC. Figure 5.5 shows an upward trend from left to right, which implies that 3DMark and Geekbench are able to capture the performance uplift that occurs with each new generation of SoC on the CPU and GPU side. Similarly, Figure 5.6 shows an upward trend of the frames per second for each benchmark of ARBench across the different generations of SoCs, which implies that ARBench is able to capture the performance improvements that occur with each new generation of SoC. Besides capturing trends that are correlated with CPU and GPU benchmarks, ARBench is able to provide other insights about the capability of existing mobile devices to execute AR workloads, which are outlined in the following evaluations.

Evaluation of existing mobile SoCs: Figure 5.6 shows the per-benchmark AR performance of different Snapdragon SoCs, as evaluated by ARBench. It shows few insights:

- AR performance of existing mobile SoCs greatly depends on the type of AR workload: figure 5.6 shows that the frames per-second have a great variation across

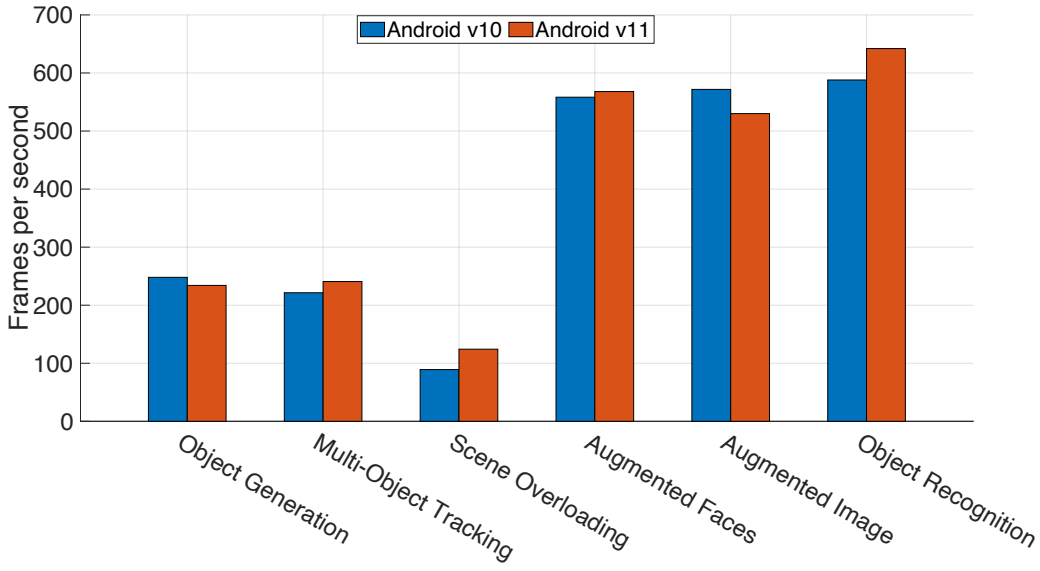

Figure 5.7: The per-benchmark AR performance of Android v10 and Android v11 while running the proposed benchmark.

different benchmarks. For instance, the frames per second vary from 90 FPS to 590 FPS for Snapdragon 855, when running the Scene Overloading and the Object Recognition benchmarks, respectively. This implies that the performance of existing commercial mobile SoCs depends greatly on the type of AR workload.

- AR workloads that involve the insertion of multiple 3D virtual objects are substantially more compute intensive than other AR workloads: comparing the performance on the first three benchmarks against the performance on the last three benchmarks on Figure 5.6, shows that AR workloads that involve the insertion of multiple 3D virtual objects in the scene are more compute intensive than AR workloads than involve the insertion of face filters, image filter and object recognition.

- The AR performance varies significantly across the different generation of SoCs: Figure 5.6 shows a substantial performance improvement across the different generations of mobile SoCs for all the ARBench benchmarks, for instance comparing the Snapdragon 632 to the Snapdragon 845, the frames per second increased from 100 FPS to 420 FPS while running the Augmented Faces benchmark.

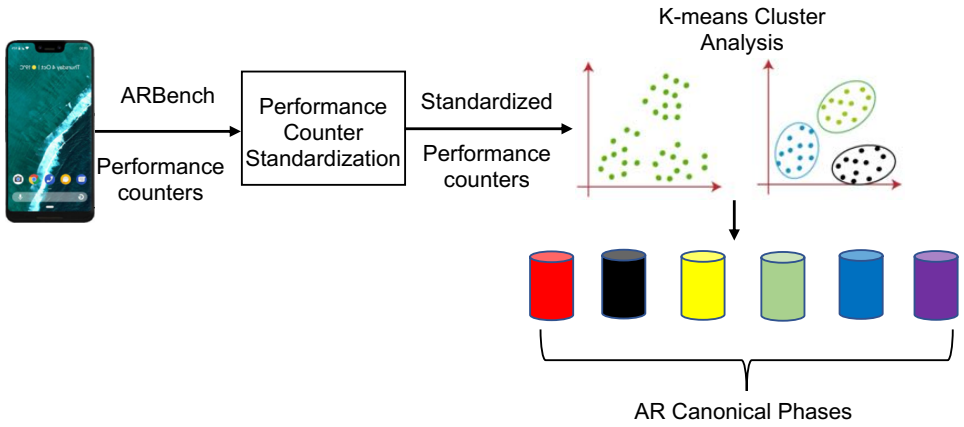

Figure 5.8: The phase analysis methodology.

Evaluation of Android operating systems: in order to identify the performance impact of different versions of the Android operating system, we run ARBench on two similar mobile devices that run on different versions of Android. Figure 5.7 shows the per-benchmark performance results for both Android v10 and Android v11. Overall, Figure 5.7 shows that the operating system version has no impact on the performance of different AR workloads. Additionally, contrasting the results of Figure 5.6 with the results of Figure 5.7, we conclude that the main bottleneck to improve the performance of AR workloads is more hardware related than operating system related.

5.6 Phase analysis of AR workloads

In this section we perform a phase analysis to identify a set of canonical phases that can be used to model and characterize the composition of the different of AR workloads. First, we introduce the approach used to perform the phase analysis, then we analyze the identified canonical phases, then we model existing AR Apps based on the canonical phases.

As shown on Figure 5.8, we start by running ARBench on the Snapdragon 865 SoC, while collecting the following performance counters on the CPU side: **instructions, branch-**

Table 5.3: The normalized average performance counter values for each AR canonical phase.

	Instructions	Branch misses	Cache misses	Cache references	Page allocation	GPU utilization
Phase 1	1.00	1.00	1.00	1.00	1.00	79 %
Phase 2	0.45	0.55	0.46	0.45	0.97	71 %
Phase 3	0.77	0.89	0.75	0.75	0.99	83 %
Phase 4	0.77	0.96	0.79	0.79	0.7	45 %

misses, cache misses and cache references, and on the GPU side we collect **page allocation and utilization**. As shown Figure 5.8, we then standardize the performance counters by subtracting the mean of each feature and dividing by the standard deviation, in order to get a mean of 0 and a standard deviation of 1. The standardized performance counters are then used to conduct a k-means cluster analysis, to identify a set of clusters that correspond to the canonical phases. In order to identify the number of clusters that reflect actual AR workload phases, the cluster analysis is performed for a different number of clusters, while keeping track of the percentage of samples of ARBench workloads that belong to each cluster. Then the number of clusters is identified by choosing the highest number of clusters possible, such that each cluster would at least represent 10% of the composition of one ARBench benchmark. This aims to prevent the creation of clusters that would represent outliers, rather than an actual canonical phase.

Table 5.3 shows the normalized average performance counter values for each identified AR canonical phase. The values in each column are normalized by dividing by the highest value, except for the last column that shows the GPU utilization. The identified canonical phases are showing four different trends on the CPU and GPU:

- Phase 1: corresponds to high CPU and GPU activity with a high number of instructions, branch-misses, cache-misses, GPU page allocations and GPU utilization.
- Phase 2: corresponds to low CPU and high GPU activity with a lower number of

Table 5.4: The canonical phase composition of the ARBench benchmarks.

	Phase 1	Phase 2	Phase 3	Phase 4
Object Generation	16 %	13 %	63 %	8 %
Multi-Object Tracking	20 %	10 %	68 %	2 %
Scene Overloading	25 %	12 %	61 %	2 %
Augmented Faces	4 %	12 %	7 %	77 %
Augmented Image	11 %	7 %	0 %	82 %
Object Recognition	2 %	14 %	0 %	84 %

instructions, branch-misses and cache-misses as compared to the other phases, with high GPU page allocations and GPU utilization.

- Phase 3: corresponds to a high CPU and GPU activity with a high number of instructions, branch-misses, cache-misses, GPU page allocations and GPU utilization. As compared to Phase 1, it has a lower number of instructions, cache-misses and cache-references, however, Phase 3 has a higher number of branch-misses per instruction as compared to Phase 1.

- Phase 4: corresponds to a high CPU and low GPU activity with a high number of instructions, branch-misses and cache-misses. On the CPU side, Phase 4 has the highest number of branch-misses per instruction as compared to Phase 1 and Phase 3, while on the GPU side, it has the lowest GPU utilization and page allocation.

Table 5.4 shows the canonical phase composition of the ARBench benchmarks, which represents the percentage of time spent on each canonical phase. As shown on Figure 5.4, the *object generation, multi-object tracking and scene overloading* benchmarks of AR-Bench involve the insertion and tracking of multiple 3D virtual objects, while augmented faces, augmented image and object recognition benchmarks are more focused on feature extraction and tracking. Table 5.4 shows that the identified canonical phases are able to detect the correlation that exists between the different ARBench benchmarks, such that the first 3 benchmarks have Phase 3 as a strong component in the canonical composi-

tion, while the last 3 benchmarks have Phase 4 as the strong component. Table 5.4 shows that the canonical compositions across Phase 1 and Phase 2 reflect another granularity of the CPU and GPU computation intensity, for instance the *scene overloading* benchmark, which has the highest rendering activity, spends 25% of the time in Phase 1, which is the phase that has the highest CPU activity combined with a high GPU activity, while the *object recognition* benchmark, which is less intensive that other benchmarks on the CPU side, spends only 2% on the CPU-GPU intensive Phase 1, and spends 14% on Phase 2, which has high GPU and low CPU activity.

In order to understand the composition of commercial AR Apps in terms of workload phases, and to verify that ARBench is composed of AR workloads that reflect existing commercial AR Apps, we perform a runtime canonical phase analysis of the commercial AR Apps shown in Table 5.1. The analysis is performed by collecting the performance counters while running the Apps on the Snapdragon 865 HDK, then each sample is classified to one of the predefined canonical phases. Figure 5.9 shows the canonical phase composition of different AR Apps. First, Figure 5.9 shows that most AR Apps are mainly composed of two phases : 1) intensive CPU-GPU phase, 2) intensive CPU and low GPU utilization phase. Additionally, Figure 5.9 shows that ARBench includes workloads that reflect the composition of existing commercial AR Apps, for instance, *Mission to Mars, Augmently, Knightfall AR, Samsung AR* and *Monster Park* are more similar to the *Object Generation, Multi-Object Tracking and Scene Overloading* benchmarks of ARBench, in terms of canonical phase composition, while *AR 3D Animal, YouCam* and *Sketch AR* are mostly similar to the *Augmented Faces, Augmented Image and Object Recognition* benchmarks of ARBench.

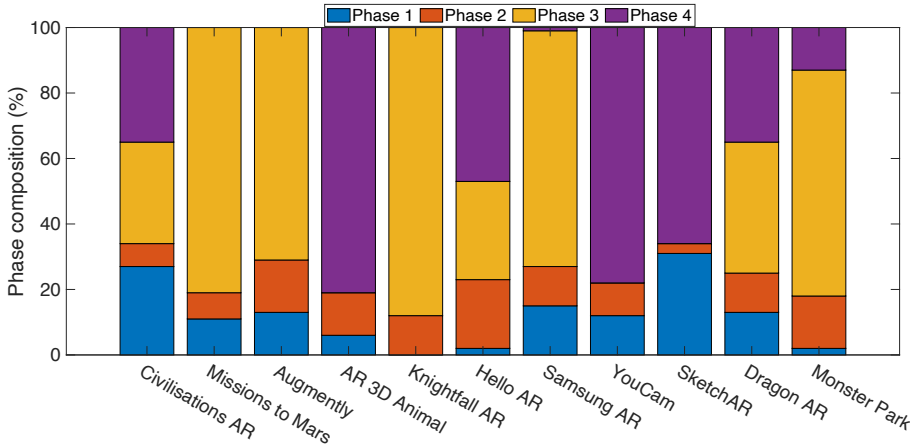

Figure 5.9: The canonical phase composition of different AR apps.

5.7 Performance and Power evaluation of different CPU multi-core configurations

In order to identify the optimal hardware configuration that could support AR workloads, we collect performance and power data for different CPU multi-core configurations. As shown on Figure 5.10, using the Snapdragon 865 SoC we test 7 different hardware configurations by turning off certain cores and capping the maximum allowed frequency on certain CPU clusters. More precisely, we test the following configurations show on Figure 5.10 :

- 4 Little + 3 Big + 1 Boost: composed of 8 different cores, divided into 4 Little cores with a maximum frequency of 1.8 GHz, 3 Big cores with a maximum frequency of 2.42 GHz and 1 Boost core with a maximum frequency of 2.84 GHz.
- 4 Little + 1 Boost: composed of 5 different cores, divided into 4 Little cores with a maximum frequency of 1.8 GHz,and 1 Boost core with a maximum frequency of 2.84 GHz.
- 4 Little + 1 Big: composed of 5 different cores, divided into 4 Little cores with a

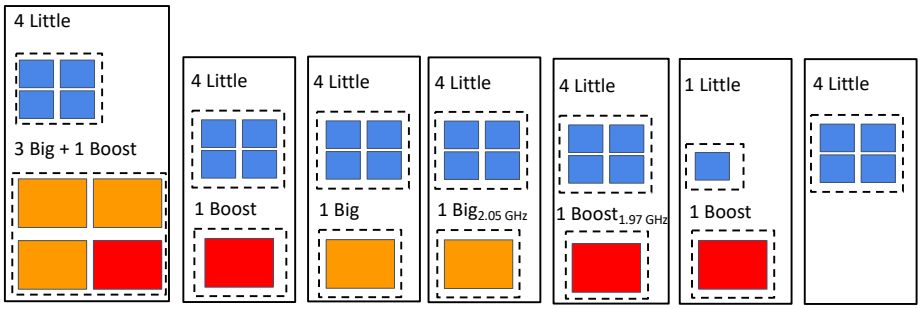

Figure 5.10: The different CPU multi-core configurations for which the performance and power trade-off is analyzed using ARBench.

maximum frequency of 1.8 GHz, 1 Big core with a maximum frequency of 2.42 GHz.

- 4 Little + 1 Big$_{2.05GHz}$: composed of 5 different cores, divided into 4 Little cores with a maximum frequency of 1.8 GHz and 1 Big core with a maximum frequency of 2.05 GHz.
- 4 Little + 1 Boost$_{1.97GHz}$: composed of 5 different cores, divided into 4 Little cores with a maximum frequency of 1.8 GHz and 1 Boost core with a maximum frequency of 1.97 GHz.
- 1 Little + 1 Boost: composed of 2 different cores, divided into 1 Little core with a maximum frequency of 1.8 GHz,and 1 Boost core with a maximum frequency of 2.84 GHz.
- 4 Little: composed of 4 Little cores with a maximum frequency of 1.8 GHz.

Figure 5.11 shows the per-benchmark AR performance of the different CPU multi-core configurations shown in Figure 5.10, as reported by ARBench. Figure 5.12 shows the total power savings for the same CPU multi-core configurations, as compared to the default case, which includes 4 Little cores, 3 Big cores, and 1 Boost core. For this evaluation, we use a performance threshold of 120 FPS, shown in a dotted line in Figure 5.11, which is the refresh rate of the state-of-the-art mobile devices. Figures 5.11 and 5.12 show that:

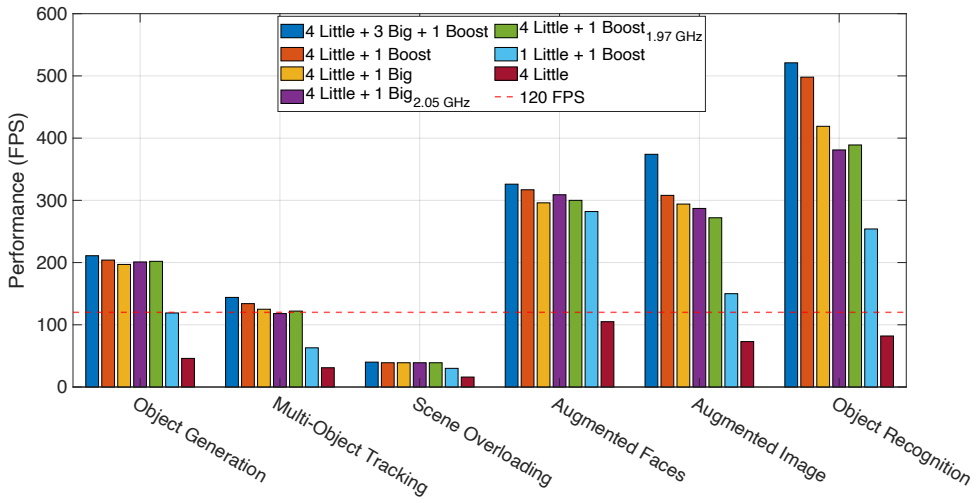

Figure 5.11: The per-benchmark AR performance of the different CPU multi-core configurations as reported by ARBench.

- All the CPU multi-core configurations fail to meet the performance requirement for the AR scene overloading workload. Even using the default configuration of a state-of-the-art SoC, the FPS for an AR workload that includes multiple 3D virtual objects could be as low as 42 FPS.

- Using only 2 cores, divided into 1 Boost and 1 Little core is sufficient to meet the performance requirements for Augmented Faces, Augmented Image and Object Recognition AR workloads, while achieving important energy savings.

- Using 5 cores, divided into 4 Little with a maximum frequency of 1.8 GHz and 1 Boost core with a maximum frequency of 1.97 GHz is sufficient to meet the performance requirements for all the AR workloads, besides the Scene Overloading benchmark, while achieving good energy savings.

- Using more than one Big core and a maximum frequency higher than 2.05 GHz is not energy efficient to run most of the AR workloads, for a target performance of 120 FPS.

- AR workloads are substantially different and there is no optimal CPU multi-core configuration for AR workloads, rather the best configuration that would achieve

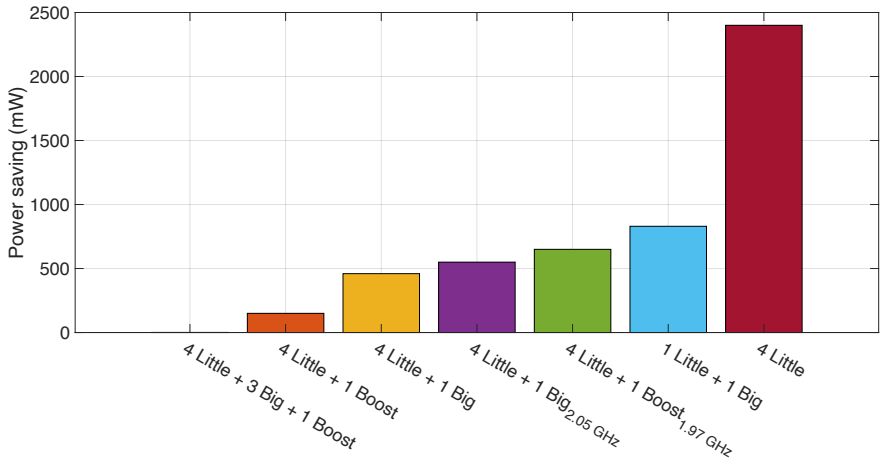

Figure 5.12: The power savings of the different CPU multi-core configurations as to the default configuration.

the performance requirements while saving on power depends on the kind of AR workload that is being executed.

5.8 Conclusion

Few challenges are facing the widespread usage of AR Apps, including a lack of a good understanding of the performance and power characteristics of existing AR Apps and commercial mobile devices when running AR Apps. In this chapter, we characterized the hardware utilization of several AR Apps using a commercial device, then we analyzed the power consumption of AR Apps per-hardware unit and per-AR process. Afterwards, we proposed ARBench, the first AR Benchmark for mobile devices. ARBench is composed of different AR workloads that reflect the existing AR Apps by stressing multiple hardware units simultaneously, and is capable of measuring the performance on each individual AR workload. The proposed benchmark is used to provide insights about the ability of existing mobile devices and Android operating systems to run AR workloads. Additionally, ARBench is used to perform a phase analysis to identify a set of AR canonical phases,

which are then used to characterize existing AR Apps. Finally, we study the performance and power trade-off of different multi-core CPU configurations and provide insights about the most efficient multi-core configuration that could run AR workloads, while meeting the performance requirements.

Chapter 6

Coordinated Self-tuning Thermal Management Controller for Mobile Devices

In this chapter we propose a coordinated self-tuning thermal management controller based on online deep learning that continuously adapts to operating conditions. Furthermore, our controller takes into account both skin temperature and junction temperature constraints in a coordinated manner. The major contributions of this chapter are as follows:

- We design a junction temperature controller that continuously tunes the PID parameters based on online learning. The controller uses a neural network that updates its weights according to operating conditions to reduce thermal violations while maximizing performance.
- We design a neural network based coordinated self-tuning thermal management controller that manages both the skin and the junction temperature by proactively

scaling the junction temperature threshold.

- We implemented a low-overhead controller on a real smartphone and we evaluate it comprehensively compared to PID [84], thermal-aware DVFS controller [48] and USTA [41] , under different ambient temperatures and workload characteristics. Our results demonstrate that our coordinated self-tuning thermal controller leads to 6% better performance, and spends up to 27× less time in thermal violation.

The rest of the chapter is organized as follows. The motivation of the proposed work is presented in Section 6.1. The proposed technique is introduced in Section 6.2. The experimental setup, the performance and thermal evaluation, as well as the neural network overhead estimation are presented under Section 6.3. Finally we summarize our conclusions in Section 6.4.

6.1 Motivation

Many thermal management techniques in the past have been developed and evaluated in servers and desktop computers. However, most of these techniques are not ideal for mobile devices, because of the different power envelopes, dimension constraints and cooling techniques. We highlight five main motivations for coordinated custom tuning of thermal management techniques for mobile devices.

1. *Workload dependent PID parameters:* Using two different workloads, SGEMM (floating point) and AES (cryptography), we tuned the PID parameters on a mobile platform based on the already explained Ziegler-Nichols approach, and it turned out that it results in different PID parameters, knowing the value of these parameters

defines the thermal behavior and performance of the controller:

$$SGEMM : K_p = 0.07\ K_i = 0.007\ K_d = 0.0018$$
$$AES : K_p = 0.03\ K_i = 0.007\ K_d = 0.0018$$

That is, we observe that the best PID controller parameter are workload dependent.

2. *Ambient temperature dependent PID parameters:* Using the same tuning approach, we tuned the PID parameters at ambient temperature 25 °C, and re-evaluated its thermal behavior at an ambient temperature 34 °C. Even if no thermal violation was observed at 25 °C, our experiments showed that the controller was unable to limit the temperature at the predefined threshold at 34 °C, since the junction temperature spent 4.5 % of the overall experiment time in a thermal violation. Thus, the behaviour of the thermal controller needs to take into consideration the ambient temperature.

3. *Multiple sources of thermal emergencies:* Mobile SoCs tend to have a more heterogeneous makeup compared to desktop/server processors, and furthermore, the whole device has to be limited because of skin temperature in addition to the junction temperature. Using thermally aggressive benchmarks, we tracked the temperatures of the big cluster, small cluster, the GPU and the skin temperature of our mobile device, and it turned out that thermal limitations arise from the the maximum junction temperature within the big cluster that reached 95 °C, as compared to 80 °C and 81 °C for the little cluster and the GPU respectively, and from the skin temperature whose temperature reached 39 °C.

4. *Fast Junction thermal transients and slow skin transients:* Our experiments showed that the thermal variation in the mobile device are much rapid and can be up to 5 °C/ms compared to only 18 °C/s in the desktop processor. Thus, mobile thermal

controllers need fast reaction times to cope with these fast transients. Despite the fast junction transients, the user skin thermal sensor typically shows very slow thermal transients that can take few minutes to ramp up to unsafe levels. Thus, we need to have different but yet coordinated mechanisms to handle both junction and skin temperatures.

5. Previously designed thermal management techniques for desktop computers or techniques demonstrated on development boards usually under-perform on mobile devices, given the different thermal constraints. we implemented the thermal-aware DVFS technique proposed by Kim et al. [48], which ended-up having 9 degrees of thermal violation, while spending 4.6% of the experiment time in a violation, compared to a regular PID that showed no thermal violation for the same performance.

6.2 Proposed self-tuning methodology for thermal management

6.2.1 Proposed self-tuning Controller for junction temperature

We first propose a self-tuning thermal controller for the junction temperature that we later, in Subsection 6.2.2, incorporate to build a coordinated controller that takes into account both the junction temperature and skin temperature. The proposed architecture for the self-tuning controller of the junction temperature is depicted in the dashed rectangle in Figure 6.1, while the secondary NN in the figure will be discussed in the next subsection. It is composed of a neural network (NN) and a regular PID controller that work together to keep the temperature under the predefined threshold. The output of the PID controller is a floating-point number, based on which it picks a DVFS level from a table that contains

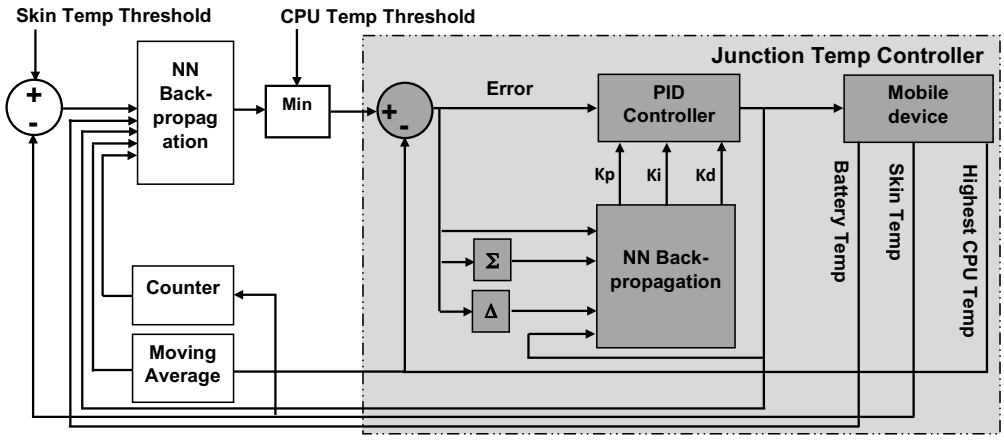

Figure 6.1: Coordinated self-tuning controller scheme to manage both the junction temperature and the skin temperature.

5 different frequencies, after rounding down the output to an integer, with 0 and 4 as minimum and maximum values, respectively. The NN takes as input four features: the current DVFS command, the thermal error, defined as the difference between the current temperature and the predefined threshold, as well as the integral and derivative errors, that bring information about how long the system spent above/below the thermal threshold and how fast is the system varying, respectively. The NN gives as output the PID parameters (i.e., K_p, T_i and T_d) that are used by the PID controller to choose a frequency level. In our work, the neural network is composed of an input layer of size 4, two hidden layers of size 8 each and 3 neurons as an output layer. The sigmoid function was used as an activation function between the hidden layers, while a linear function was used at the output.

After applying the DVFS command the controller waits for 10 ms, then measures the new thermal error, integral and derivative errors, which are then used by the NN to update the weights using back-propagation and the absolute error between the target gains and the current outputs of the NN as a loss function. The target gains K_p^{target}, K_i^{target} and K_d^{target} represent the gains that the NN should aim for the next time it encounters the current features, and they are defined according to the following equations:

$$K_p^{target} = K_p^{current} + R_p e(t)$$
$$K_i^{target} = K_i^{current} + R_i \int_0^t e(t')dt'$$
$$K_d^{target} = K_d^{current} + R_d \frac{de(t)}{dt}$$

The neural networks is trained on the fly based on target gains to iteratively reduce the error, to get to the optimal gains based on the corresponding features. R_p, R_i and R_d here stand for hyper-parameters that define the rate at which we move towards the optimal parameters, similar to the learning rate of the NN which defines the rate at which the weights are adjusted in regard of the loss. Each gain is updated proportionally to its respective error, where the absolute error is computed between the three outputs of the NN and their respective target gain, because at time t, the current PID parameters are not necessarily equally distant from their respective optimal gains.

The self-tuning controller has the advantage of being able to tune the gains based on the current workload, and as a result, it picks the optimal frequency level that uses the thermal available headroom efficiently, ramping DVFS levels in a faster and more effective way, resulting in performance improvements compared to a regular PID, which has constant gains.

6.2.2 Proposed coordinated junction & body self-tuning controller

Our thermal analysis in Section 6.1 has shown that the junction temperature, as well as the skin temperature are the source of the thermal limitations. The skin temperature might highly affect the user experience, because at a certain temperature level it creates unpleas-

ant burning sensations, and it is usually the result of an accumulated heat coming out from the SoC, which motivates us to design a coordinated self-tuning controller to manage both the junction and the skin temperature. The proposed scheme for such a coordinated controller is depicted in Figure 6.1, where the main idea here is to build up on the previously proposed self-tuning controller, by adding a secondary NN that continuously adjusts the maximum allowed thermal junction threshold giving to the junction self-tuning controller. The NN architecture takes as five inputs: (1) the error between the current skin temperature and the maximum allowed skin temperature, (2) battery temperature, since it is highly correlated with the skin temperature, and (3) the current frequency level since it is correlated with the future temperature levels. As noted earlier in Section 6.1 the skin temperature has slow transients, the temperature of which is affected by the accumulated heat across the whole device, as result, two other features are used as input to the NN: (4) the time spent on the current skin temperature level, which is computed using the counter in the scheme, and (5) a moving average of the highest CPU temperature. We define the moving temperature average as:

$$Temp\_MA[n] = \alpha Temp[n] + (1 - \alpha)Temp\_MA[n - 1], \tag{6.1}$$

where $Temp\_MA[n]$ stands for the temperature moving average at time n, $Temp[n]$ is the maximum CPU temperature at time n and α is the forgetting factor, which determines how much weight past data is given. The output of the NN is the junction temperature threshold that would ensure that the skin temperature is below the threshold, while the minimum function in the scheme, ensures that neither the skin nor the junction temperature threshold are violated.

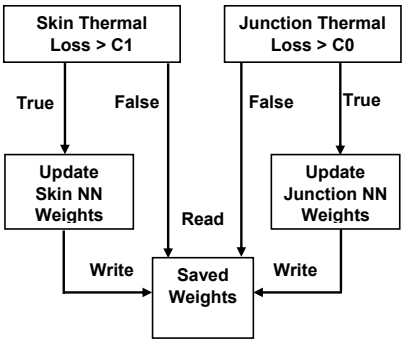

Figure 6.2: Neural Network Update Mechanism.

6.2.3 Efficient online Learning

Our controllers make use of both offline and online learning phases. In the offline training phases, the network is trained for the first time while running some workloads. The learning process ends as soon as the skin and junction thermal loss are below the predefined thresholds. During the online phase, for efficiency reasons the neural network weights are only updated when needed. Figure 6.2 shows the update mechanism that relies on continuously computing the skin thermal loss and the junction thermal loss, representing the time the corresponding temperature spends above the predefined thresholds over the previous 10 seconds. Both the skin and junction thermal losses are compared to two constants C0 and C1, defined experimentally, such that if the loss is higher than the predefined thresholds, the corresponding weights are updated, otherwise, the controller uses the saved Neural Network weights. We quantify the overheads of online learning in Section 6.3 and show it leads to net improvement in performance.

6.3 Experiments and results

6.3.1 Experimental setup

All the experiments are performed on an Android-based (Oreo version 8.0.0) Google Pixel 2 XL phone, since its Snapdragon-835 octa-core processor which is based on the ARM big.LITTLE architecture is actually used in more than 20 different phone models.

The performance is measured based on the Geekbench multi-core score. The Geekbench score is based on the run-time of each workload, the lower the run-time the higher the score is. Geekbench was chosen because it is the most widely used benchmark in the mobile market by both companies and end-users. The benchmark score is based on 25 workloads of different categories, including cryptography, integer, floating-point and memory workloads. Geekbench reports as well the score per workload category, knowing that each one of the categories consists of several workloads. As discussed in Section 6.1 we only focus on the big cluster maximum junction temperature and the skin temperature, since they are the source of the thermal violations based. The sampling rate of the data collection is 2.5 ms to ensure we can control the fast junction thermal transients.

For temperature thresholds, we use a maximum of 70 °C for the junction temperature and a body thermal threshold of 42 °C. Note that we do not use the highest possible maximum values to avoid the triggering of the built in threshold mechanisms within the SoC and the Android OS. We perform some of our experiments at different ambient temperatures using the thermal chamber TestEquity TEC1.

	Max Temp	Time spent in violation
Thermal-aware DVFS [48]	83 °C	4.63 %
Regular PID	69 °C	0 %
Self-tuning PID	70 °C	0 %

Table 6.1: Thermal evaluation of the implemented junction temperature controllers.

6.3.2 Results

Regular PID: The PID regulator was implemented based on Ziegler-Nichols tuning method as described earlier in Chapter 2. To reach the target steady state, a stable workload with heavy floating point calculations was used to avoid thermal variations inherent to the workload. Based on the closed loop system, K_p was incremented until a steady oscillation state was reached. The obtained values are as follows: $K_p = 0.18 \quad K_i = 0.0125 \quad K_d = 0.003125$. After tuning the controller with these parameters, to further validate the PID regulator, we implemented the thermal-aware DVFS technique proposed by Kim *et al.* [48].

As shown in Table 6.1 running the entire Geekbench 25 benchmarks at an ambient temperature of 25 °C showed that for both the PID and the self-tuning controller the temperature never exceeds the target 70 °C threshold, while the technique in [48] spent 4.6% in a thermal violation, with the maximum temperature reaching 83 °C, this underperformance is due to the fact that this technique was not evaluated on a real smartphone platform, which has much faster junction thermal transient, as previously mentioned in subsection 6.1. For this reason, the thermal-aware DVFS technique [48] is not considered for next comparisons, which are achieved at higher ambient temperature.

Coordinated self-tuning junction & body controller: The controller was implemented using the C Language to ensure the best efficiency while minimizing the overhead. The most critical parameters that needed to be tuned are the R_p, R_i and R_d hyper-

parameters and the size of the NN. The R parameters were determined by running the controller while keeping track of the target gains, such that if the values keep just increasing or take a long time to converge then the corresponding hyper-parameter is increased, otherwise, if the gains keep oscillating between two values then the corresponding hyper-parameter is decreased. The size of the neural network was defined by simply observing the performance of the overall controller while gradually increasing the depth of the neural network. The controller is engaged only each 10ms, because the online learning feature of the controller makes it proactive, and such sampling rate prevents an undesirable high number DVFS transitions. The results show that only within 3 runs (6 minutes of learning overall) the network already learned to tune the PID parameters according to the different features, the thermal violation went down from 35% on the first run to 0.07% on the third run, while bringing 5% performance increase compared to regular PID. This performance increase results from the ability of our controller to timely apply the highest DVFS command under the current thermal conditions by tuning the PID parameters; the collected data show that these parameters keep tuning throughout the experiment, adapting to the load and thermal status of the SoC, rather than being just constant numbers like in regular PID controller. It is interesting to note that the PID parameters have spikes that correspond to the beginning of execution of a different workload. These tuning spikes are due to the fact that the beginning of a new workload usually leads to the highest temperature variation; thus, the NN continuously adapts the parameters at early phases until steady-state is reached.

This continuous self-tuning nature of our controller enables it to cope with changes in ambient temperature. Using our thermal chamber, thermal and performance evaluation experiments were performed at higher ambient temperatures (34 °C) to show the advantage of adaptive PID parameters over static parameters at different ambient temperatures and for different workloads. It turned out that the regular PID is unable to keep the temperature

below the thermal threshold, even though it was able to perfectly keep the temperature under the threshold at lower ambient temperatures. The regular PID spent around 22 seconds in a thermal violation, whereas the self-tuning controller spent only 792 millisecond in a thermal violation, which is 27 times less than the regular PID. The self-tuning controller was able to adapt to the new thermal conditions by updating the weights, and as result, it gives dramatically less thermal violations.

To further demonstrate the superiority of our controller, we compare it to two other approaches:

1. **USTA [41]:** Using the methodology recommended by the authors [41], the decision tree model was firstly derived offline based on the collected data that consists of a moving average of 20 samples of the following features: the CPU temperature, the frequency and the battery temperature. The derived model that had a maximum number of splits equal to 10, has shown a comparable accuracy to the original work [41] with an average error equal to 0.15 °C. Afterwards, the prediction model, as well as the already explained mechanism were implemented in C on the phone. This controller is designed to manage the skin temperature.

2. **USTA + PID:** The proposed approach is compared as well to the combination of both (USTA+PID), where we run both techniques at the same time, to manage the skin and junction temperature.

We report the results in Table 6.2 broken down by Geekbench workload type: cryptography, integer, floating points and memory workloads. Each one of these benchmarks was run for 20 minutes while putting the phone at an ambient temperature of 34 °C using the thermal chamber. The reported performance on the table is the percentage increase of the category Geekbench score compared to the score of (USTA+PID), which is used as

Controller	Performance					Thermal		
	Crypto	Integer	Float	Memory	Total	Skin	Junc	Max
Our work	2.4%	4%	8%	17%	6%	10%	1.5%	73.9
PID	3.6%	1%	5.9%	15%	3.5%	66%	4.5%	75.5
USTA[41]	2.2%	1%	1%	−5%	0.7%	14%	2.7%	77
USTA+PID	0%	0%	0%	0%	0%	13%	1.9%	74.2

Table 6.2: Performance and thermal evaluation of the implemented controllers.

the baseline. Adding to the performance evaluation, the thermal management efficiency is evaluated through three metrics: the percentage time of skin and junction violation, throughout the experiment, and the maximum reported junction temperature.

In the following the thermal and performance evaluation:

1. **Thermal :** Our work shows a significant improvement compared to PID and to USTA, with only 10% of the time spent in a skin temperature violation, compared to 66% for PID, with a maximum junction temperature of 73.9 °C compared to 77 °C for USTA. This shows the advantage of the adaptivity aspect that the proposed technique offers through neural networks, in contrast to constant values for the other techniques, resulting in poor performance at higher ambient temperature levels.

2. **Performance :** our technique brought up to 8% performance improvement on floating point workloads, 17% on memory workloads and 6% improvement over the 25 benchmarks compared to USTA+PID, which actually was able to perform thermally better than the PID regulator and the USTA running separately, but it performed poorly in terms of performance, which actually shows the importance of a coordinated management of the skin and junction temperature, that our work offers.

Keeping track of the PID parameters while running Geekbench multi-core showed, that the PID parameters keep varying throughout the experiment, adapting to the load and

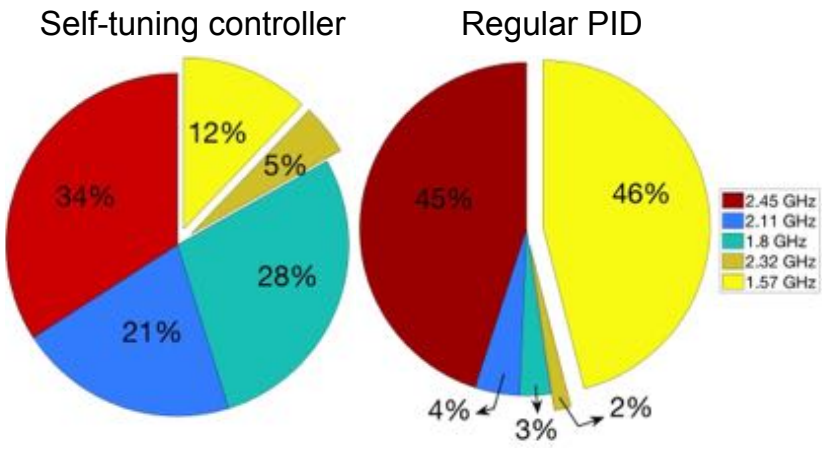

Figure 6.3: Frequency distribution of the self-tuning controller compared to the regular PID while running Geekbench 25 benchmarks.

thermal status of the SoC. The proposed technique improved the performance, while having a lower thermal violation because the neural networks learned a conservative frequency scaling technique as shown in Figure 6.3, that tends to less frequently pick the highest frequency level, where it sacrificed 11% of the time spent on the highest frequency compared to the regular PID, but gets the advantage of generating less heat, and spending less time at the lowest frequency level, by spending only 12% compared to the regular PID that ended-up spending 46% of the time at the lowest frequency level.

Overhead of the self-tuning controller: In order to measure the run-time overhead, the PID controller and the NN were set to run all the required computation on the mobile device without applying the DVFS decisions, which allows to quantify only the overhead coming from the computation. By comparing the scores of the three runs, that actually reflect the run-time, for the case where the NN is performing computation, and three runs to the case where Geekbench is executing without the NN overhead, it turned out that the overhead is less than 1%, which is within the GB noise range. Furthermore, *all of our earlier reported performance results already incorporate the impact of the overhead.*

6.4 Conclusion

In this work, a thermal analysis was conducted, through which we spotted the sources of the thermal limitations, and motivated the need to devise thermal management techniques that are relevant to the mobile devices requirements. Then we designed a coordinated self-tuning thermal management controller based on online deep learning, that keeps varying the PID parameters to adapt to different running conditions, and ensures both junction and skin temperature management in a coordinated fashion. The controller was implemented on a real smart-phone and evaluated comprehensively under different ambient temperatures and workload characteristics. We showed the advantage of an adaptive and coordinated thermal management by comparing our technique to other implemented techniques from the literature. The results show that the proposed approach achieves better performance and thermal management, with up to 6% performance increase across 25 benchmarks, while spending up to 27$\times$ less time in thermal violation for the junction temperature controller.

Chapter 7

CasCon: Cascaded Thermal And Electrical Current Throttling for Mobile Devices

In this chapter we propose a cascaded controller for mobile devices that controls the different sources of current and thermal emergencies in a coordinated manner. The major contributions of this chapter are as follows:

- We design a cascaded controller (CasCon) that manages the skin temperature, the junction temperature and the electrical current in a coordinated manner. In contrast to existing work that tackles these issues separately, leading to a suboptimal control.
- In contrast to existing work, the proposed controller dynamically changes the thermal and electrical current caps in runtime, allowing it to save power and improve performance. We also introduce a frequency locking function that significantly reduces the number of DVFS transitions, hence bringing extra power savings [48].

- We implement our CasCon controller on a real smartphone and we evaluate it comprehensively at different ambient temperatures. Our results show that the proposed controller successfully prevents current and thermal violations even at high ambient temperatures, while bringing up to 6.5% performance improvements and 18% power savings compared to previous work.

The chapter is organized as follows: Section 7.1 motivates the proposed work. Section 7.2 describes the proposed work. Section 7.3 presents the results and the experimental setup. Section 7.4 concludes the chapter.

7.1 Motivation

Many existing power and thermal management techniques have been evaluated for servers, desktop computers or development boards [76, 64, 34]. However, existing techniques dissociate the current throttling, the junction temperature throttling and the skin temperature throttling by controlling each measure separately [31, 48, 41]. Even though, these throttling techniques rely on the same underlying mechanisms (e.g., DVFS) to achieve the throttling, and as a result, sub-optimal or even conflicting decisions can occur. In addition, most techniques [76, 64, 34] are evaluated either through simulation, or through the use of desktop/server processors or development boards that have different thermal and power characteristics from mobile devices. Furthermore, most techniques usually evaluate their approaches either at a fixed ambient temperature or they neglect its effect [76, 64, 79], despite it has a strong impact on skin temperature.

We describe three motivations for coordinated thermal and electrical current throttling for mobile devices:

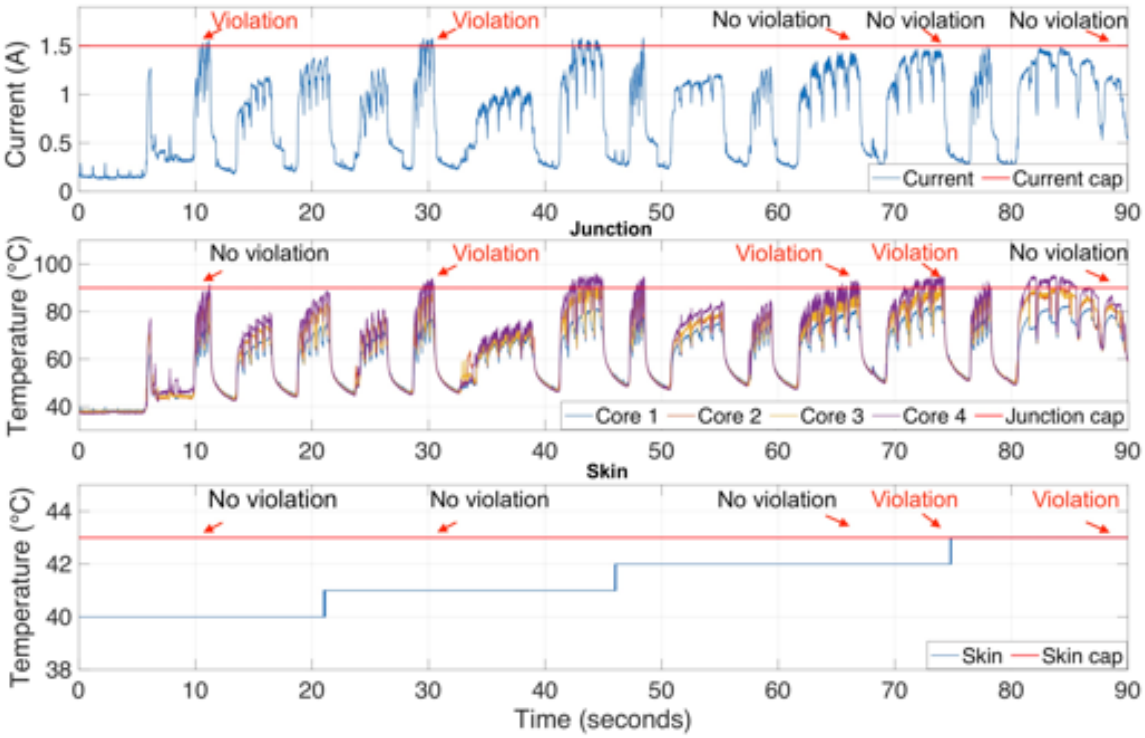

Figure 7.1: The electrical current and thermal traces while running Geekbench on the Google 2 XL.

1. Current and thermal violations can occur separately and jointly: By running Geekbench multi-core on the Google Pixel 2 XL mobile phone, we show in Figure 7.1, that electrical current, junction and skin temperature violations can occur separately, as they can occur jointly at the same time. This raises the need to design a controller that handles all these violations in a coordinated manner by capturing the physical relationships that exist among the different measures.

2. Multiple sources of current and thermal emergencies: The mobile Li-ion batteries maximum continuous discharge and instantaneous discharge current should be limited at 1C and 3C[1], respectively [82]. Adding to the current limit, our experiments using CPU and GPU workloads show that the big CPU cluster leads to thermal violations by reaching 95°C, while the temperature of the GPU and little CPU cluster never exceeds 80°C. Thus the thermal limitations arise from the the maximum junction temperature within the big

[1]A 1C rate is defined as the discharge current that discharges the entire battery in 1 hour, that is 1× and 3× of its mAh rating.

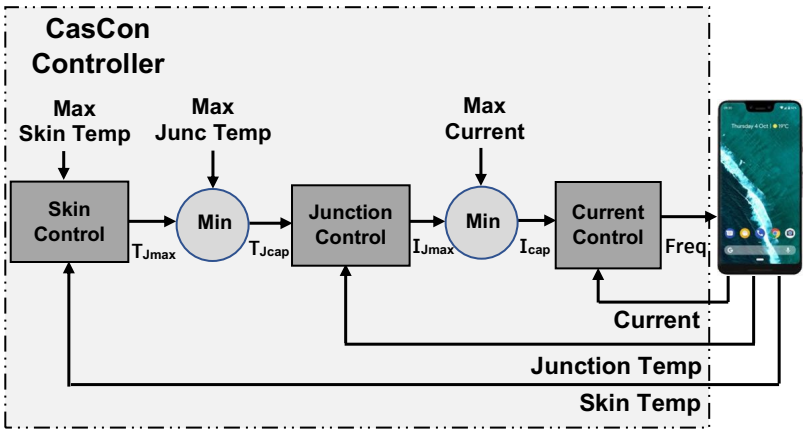

Figure 7.2: Proposed coordinated thermal and electrical current controller.

CPU cluster of the SoC and the skin temperature.

3. Fast junction thermal transients, and slow skin transient: The thermal transient on mobile SoCs can be significantly faster than on desktop processors. These fast transients can lead to high DVFS transitions, which can affect the performance and the power consumption [48, 31]. Despite the fast junction transients, the skin temperature typically shows very slow thermal transients that can take few minutes to ramp up to unsafe levels. Thus, we need different but yet coordinated mechanisms to handle both fast and slow transients to maximize performance while saving power.

7.2 Proposed CasCon controller

The design of the proposed controller depicted in Figure 7.2 is inspired from the physical relationships that exist among the different measures, such that the control is driven from the fastest time transient measure to the slowest in a coordinated manner through a cascaded control. The skin controller defines a *maximum junction temperature* T_{Jmax}, since the increase in skin temperature is due to the heat accumulation arising from high junction temperatures over time. The minimum between T_{Jmax} and the maximum junction tem-

perature allowed physically defines the *junction cap* T_{Jcap} given as input to the junction controller. The junction controller defines in its turn a *maximum current* I_{Jmax}, since the increase in junction temperature is due to power dissipation arising from high electrical current. Finally, the *electrical current cap* I_{cap} defined by the minimum between I_{Jmax} and the maximum current indicated by the battery specifications is given as input to the current controller that scales the DVFS settings.

7.2.1 The electrical current controller

The current controller takes as input I_{cap} and the current values of frequency and electrical current. The controller outputs a target frequency that allows to cap the electrical current at the predefined maximum level while maximizing performance. This is achieved through two steps:

1. Step 1: The electrical current depends on the utilization, the voltage/frequency settings and the temperature. However, only the utilization and the voltage/frequency settings are responsible for the abrupt changes of the current. Thus, we use a current-frequency function $I2F(\cdot)$ to find the optimal target frequency, such that the current does not exceed I_{cap}. Because the utilization is linearly related to the power/current, the current-frequency function $I2F(\cdot)$ is computed at a fixed utilization U_{ref}, by running a single-phase workload at multiple frequencies, while measuring the corresponding electrical current. Then the target frequency F_{target} at time t is computed based on the current utilization $U(t)$, as follows:

$$F_{target}(t) = \frac{U_{ref} * I2F(I_{cap})}{U(t)} \tag{7.1}$$

2. Step 2: In order to reduce the number of DVFS transitions for performance and

power efficiency considerations, we introduce a *frequency locking function*. This function forces the CPU to spend a minimum amount of time w at the current frequency level F_t, before increasing it to F_{target}. However, the frequency is decreased without considering the time window w if F_{target} is less than the current frequency.

7.2.2 The junction temperature controller

The increase in average electrical current leads to a higher junction temperature, due to power dissipation in computing units. As result, as shown in Figure 7.2, the thermal controller is cascaded with the current controller. It computes I_{Jmax}, which represents the maximum electrical current that will not lead to a junction temperature violation, using a PID control:

$$I_{Jmax}(t) = K_p e(t) + K_i \int_0^t e(t')dt' + K_d \frac{de(t)}{dt}, \tag{7.2}$$

where K_p , K_i ,K_d are the PID gains, while $e(t)$ represents the error between the current junction temperature and the junction temperature cap. The proposed junction controller is different than a regular thermal PID controller, since it prevents thermal violations by capping the maximum electrical current rather than directly controlling the frequency. The PID parameters are tuned using Zigler-Nichols method [84].

7.2.3 The skin temperature controller

The increase in the junction temperature leads to heat accumulation inside the entirety of the mobile device, which increases the skin temperature. As result, as shown in Figure

7.2, the skin controller is cascaded with the junction controller. The skin violations are prevented by decreasing the maximum allowed junction temperature proportionally to the amount of time it spends 1°C away from the skin cap as follows:

$$T_{Jmax}(t) = T_{Jmax}(t_0) - \beta * (t - t_0), \tag{7.3}$$

where $[t_0, t]$ represents the time range spent at a skin temperature that is 1°C away from the cap, and β is a constant defined experimentally, which determines the speed of the junction temperature reduction. The intuition behind Equation 7.3, is that the skin temperature has a slow transient, and the more time it spends at a certain level, the more likely for the temperature to increase to the next level.

7.3 Results and Experimental Setup

7.3.1 Experimental setup

All the experiments are performed on an Android-based (Oreo version 8.0.0) Google Pixel 2 XL phone, that comes with the Snapdragon-835 SoC. To generalize our results for more than one smartphone model, we specifically chose the previously mentioned SoC because it is used in more than 20 different smartphone models. *Performance* is measured using 25 benchmarks of different categories, including cryptography, integer, floating point and memory workloads from the Geekbench set. The Geekbench score is based on the runtime of each benchmark, the lower the time, the higher the score is. The thermal and frequency data is collected through the phone internal sensors using a C program. The CPU current is measured using a model built using regression analysis, by collecting the total current

using a power monitor while running CPU-workloads.

As discussed, the CPU is the only source for thermal throttling, so our controller scales the DVFS settings of the CPU. The $I2F(\cdot)$ function, the PID and the β parameters are workload independent, so they need to be determined only once. The controller was implemented using a C program, which reads the different sensor values and scales the frequency each 10 ms using the android system nodes. The controller has less than 1% run-time overhead.

It should be noted that it is not feasible to compare to the default phone governors, since such comparison requires disabling the default governors to assess our controller, and this is not possible without access to confidential information that is not publicly available. As result, we compare to **Governor 1** and **Governor 2** that are based on current and thermal controllers from the literature. **Governor 1** uses a PID controller [84] for current control and uses the frequency stabilization technique introduced in [48] for junction control. **Governor 2** uses a PID controller [84] for both current and junction temperature control. The skin temperature controller of both governors is based on USTA [41].

To avoid the interference with the built-in phone controllers, the current and temperature caps are set to lower values than the permitted ones to avoid any triggering of the default phone governors. We use 1 A for current cap, we use 78°C for the junction cap and 42°C for the skin cap. The experiments are performed at different ambient temperatures using the thermal chamber TestEquity TEC1.

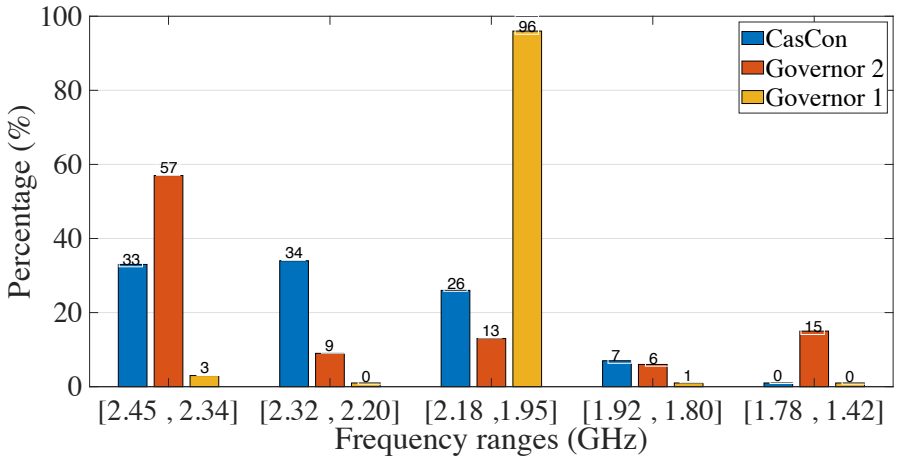

Figure 7.3: Time spent on each frequency range by the different governors.

7.3.2 Results

The proposed controller is evaluated by running the 25 multi-threaded workloads of Geekbench, first at an ambient temperature of 25 °C. Afterwards the evaluation is performed at an ambient temperature of 35 °C, which is representative of hot summer conditions, using the thermal chamber.

1. Evaluation of the coordinated thermal and electrical current controller at 25 °C: Figure 7.3 shows the time spent on each frequency range by the different governors. It shows that CasCon picked the frequency levels to create a lasting performance, rather than just blindly trying to run at the highest frequency levels. For instance, Governor 2 tried to maximize the time spent at high frequencies by spending 57% of the time in the frequency range of [2.45 GHz, 2.34 GHz], however, due to thermal and current violations this lead to 15% of the time spent at the lowest frequency range, compared to 0% for CasCon, which gave our governor a substantial performance advantage.

As given in Table 7.1, our work has a significant performance improvement compared to Governors 1 and 2, based on the Geekbench scores that actually reflect the runtime.

Table 7.1: Performance, thermal and electrical current evaluation of our controller at 25 °C.

Controller	Performance					Current, thermal and power			
	Crypto (%)	Integer (%)	Float (%)	Memory (%)	Total (%)	N I_{viol}	T_{Jviol} (%)	DVFS trans	Power (%)
CasCon	14	5.7	7.6	0	6.5	0	1	127	-18
Governor 1	5.3	6.4	-7.1	-24	-7.9	1	1	243	-20
Governor 2	0	0	0	0	0	8	1	8185	0

Table 7.2: Performance, thermal and electrical current evaluation of our controller at 35 °C.

Controller	Performance					Current, thermal and power			
	Crypto (%)	Integer (%)	Float (%)	Memory (%)	Total (%)	N I_{viol}	T_{Jviol} (%)	T_{Sviol} (%)	Power (%)
CasCon	3.7	3.2	6.6	6.3	4.3	1	<1	0	-4.4
Governor 1	-8.3	-9.3	-14.8	-9.7	-10.6	2295	6.6	3.3	-1.3
Governor 2	0	0	0	0	0	2607	8	0	0

As compared to Governor 2, it shows a performance improvement on most of the sections of Geekbench with an overall performance improvement of 6.5%. Furthermore, our controller shows a significant decrease in the number of DVFS transitions, which leads to 18% power savings compared to Governor 2.

2. Evaluation of the coordinated thermal and electrical current controller at 35 °C: Figure 7.4 shows the electrical current trace, the skin temperature, and the temperature values of the four cores of the big cluster, which correspond to the junction temperature. The figure shows that the cascaded control helps in efficiently coordinating between the different sources of emergency to prevent any sustained electrical current, junction temperature or skin temperature violation:

- In the time range [10s, 75s], the controller was able to prevent several junction temperature violations by dynamically changing the current cap.
- In the time range [245s, 305s] the coordination taking place between the skin tem-

perature, junction temperature and electrical current controllers is shown more clearly. The skin temperature spent more than 50 seconds at 42 °C without getting to a thermal violation, due to the gradual decrease of the junction cap, which then leads to a gradual decrease of the electrical current cap. Thus, preventing skin violation through cascaded control.

Table 7.2 shows that our technique brought a performance improvement on most of the sections of Geekbench, with an overall performance improvement of 4.3% and 4.4% power savings. The performance improvement seems to be less at 35 °C, because Governor 1 and 2 had many thermal violations, allowing them to use a higher thermal budget than our governor, which made the performance improvement seem less important as compared to 25 °C. Most importantly for this experiment, the proposed controller was able to prevent current, junction and skin temperature violations even when the ambient temperature increased by 10 °C. On the other hand, Governors 1 and 2 were unable to prevent current and thermal violations, as several violations took place.

7.4 Conclusion

In this work, we designed a cascaded thermal and electrical current controller, that manages the current, the junction and the skin temperature in a coordinated manner, by capturing the physical relations that exist among the different emergency sources. The controller was implemented on a real smartphone and evaluated against different management techniques from the literature at different ambient temperatures. The proposed controller, CasCon, achieves 6.5% performance improvement and 18% power savings compared to existing techniques, while successfully preventing current, temperature violations and substantially decreasing DVFS transitions.

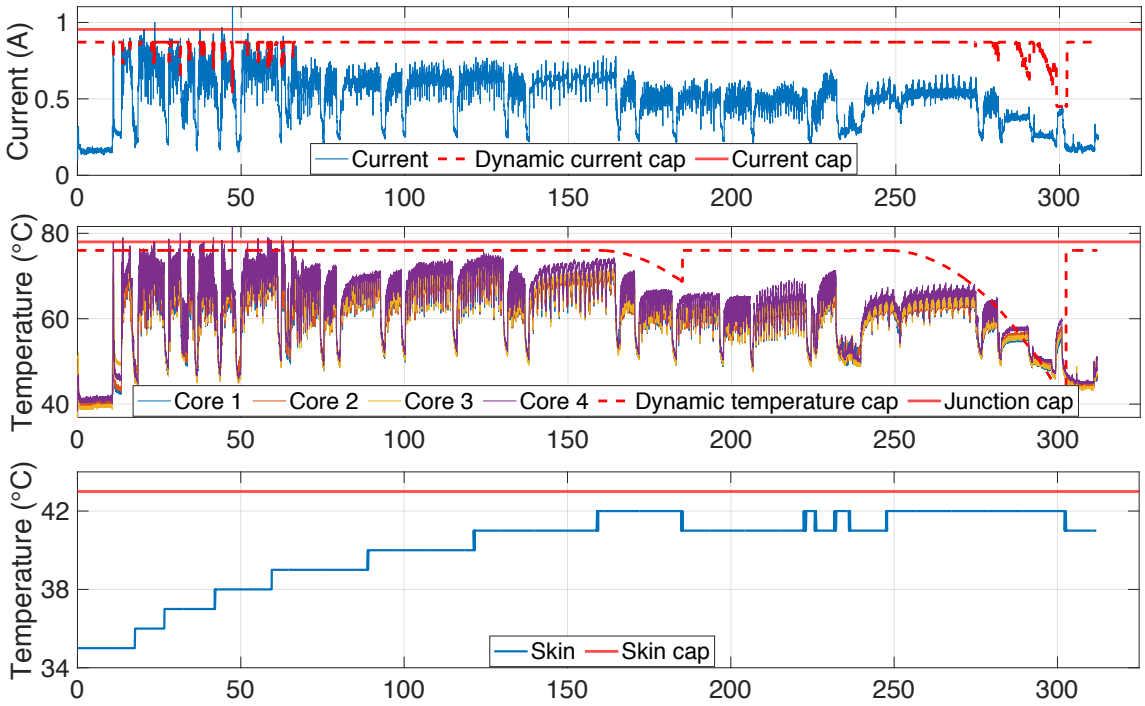

Figure 7.4: CasCon : run-time dynamic electrical current and temperature capping.

Chapter 8

Workload- and User-aware Battery Lifetime Management for Mobile SoCs

In this chapter, we propose a novel workload- and user-aware battery lifetime management that maximizes the performance under the user's desired battery lifetime. Our approach leverages insights about the running workloads by collecting CPU-GPU performance counters, which are used to proactively scale the CPU-GPU frequencies using machine learning. Additionally, to enable the user-awareness we design a model that predicts energy consumption based on the user usage history.

To summarize, the contributions of this chapter are as follows:

- We design the first workload- and user-aware battery lifetime management technique. The workload-awareness is achieved through performance counters, while the user awareness is incorporated by representing the user-usage history through a set of canonical phases (CP).

- We propose a novel model that predicts the energy consumption based on the user usage history. This model makes the proposed technique user-aware, and helps in better meeting the user's desired lifetime.
- The proposed battery lifetime management is achieved by scaling both the CPU and the GPU DVFS levels, unlike previous techniques that do not consider the GPU.
- We implement our technique on a commercial smartphone and compare its performance against state-of-the-art battery management techniques. We show that our technique achieves 15.8% and 9.4% *QoS* improvement on the CPU and GPU, respectively, while meeting the lifetime target and decreasing the *QoS variation* by 10x.

The chapter is organized as follows: Section 8.1 motivates the proposed work. Section 8.2 describes our proposed technique. Section 8.3 presents the evaluation results of our technique compared against state-of-the-art techniques. Section 8.4 concludes the chapter.

8.1 Motivation

The optimal user experience for mobile devices depends on providing a seamless performance until the next battery recharge. However, mobile devices are used differently across users. Furthermore, an important portion of the power consumption and performance in the new generation of mobile devices depends on other computing units than the CPU. Hence, we highlight four main motivations for the proposed work.

Maximizing QoS under target battery lifetime: A recent study has shown that a considerable portion of the recharges are driven by context (e.g. time, location,etc.) and there

is a great variation among users in exploiting the available battery charge [24]. Hence, to provide the best balance between QoS and battery lifetime, a management technique should consider the user desired target battery lifetime. Hence, the problem formulation of this work aims at maximizing the QoS, which is about minimizing workload runtime for the CPU and maximizing the FPS for the GPU, given a desired target battery lifetime.

High QoS variation: We implement several battery lifetime management techniques, namely, Powersave [42], BUSQ1 and BUSQ3 [51]. By evaluating the QoS variation we show that the performance variation can be as high as 60%, which greatly affects the user experience. Thus, we need to devise a management technique that provides a smooth experience to the user.

Workload-aware management: By running different workloads on the Google Pixel 2 XL at 2.45 GHz, 2.20 GHz and 2.11 GHz, we show that the decrease in performance caused by the frequency downscaling highly varies based on the workload. The performance decrease ranges from 2% to 10% for 2.20 GHz, and 2% to 15% for 2.11 GHz, as compared to the performance at 2.45 GHz. Thus, a workload-aware management would allow to save power when the frequency is over-scaled, and to improve performance when it is under-scaled.

Cooperative CPU-GPU management: Running 3DMark as a GPU workload and Geekbench as a CPU workload on a commercial smartphone, we show that the GPU has a higher power profile with an average power of 6.8 Watts, as compared to 5.5 Watts for the CPU workload. Thus, it is crucial to include both the CPU and the GPU in the management technique.

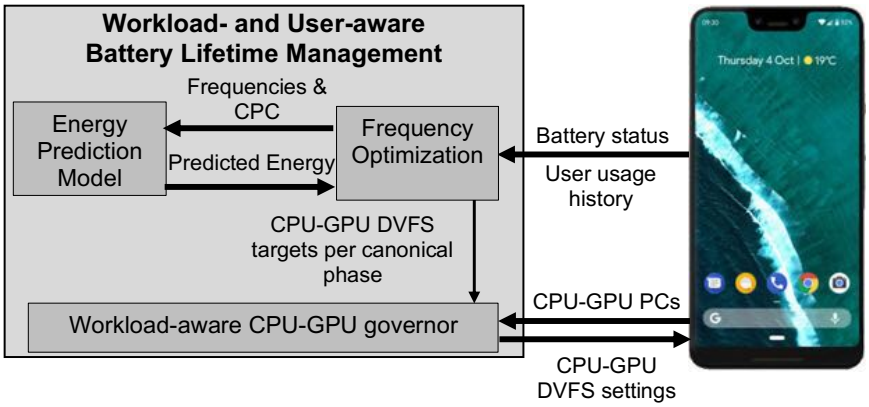

Figure 8.1: Workload- and User-aware Battery Lifetime Management.

8.2 Proposed work

The proposed battery lifetime management technique, which is depicted in Figure 8.1, uses frequency optimization and an energy prediction model to find the optimal CPU-GPU frequency targets per canonical phase (CP). The workload-aware governor continuously collects performance counters (PC), and classifies each sample as one of the CP, then the frequency is set to the frequency target corresponding to this CP, such that QoS is optimized while meeting the target battery lifetime. In the following subsections, we describe the main blocks of the proposed work, namely: A) the workload-aware governor, B) the energy prediction model and C) the frequency optimization.

8.2.1 Workload-aware governor

The workload-aware governor (WLA) shown in Figure 8.2, is designed through an offline analysis, during which a set of CPs are identified through cluster analysis. Then in run-time, the workload-aware governor is used with the frequency optimization and the energy prediction model to choose the optimal DVFS settings.

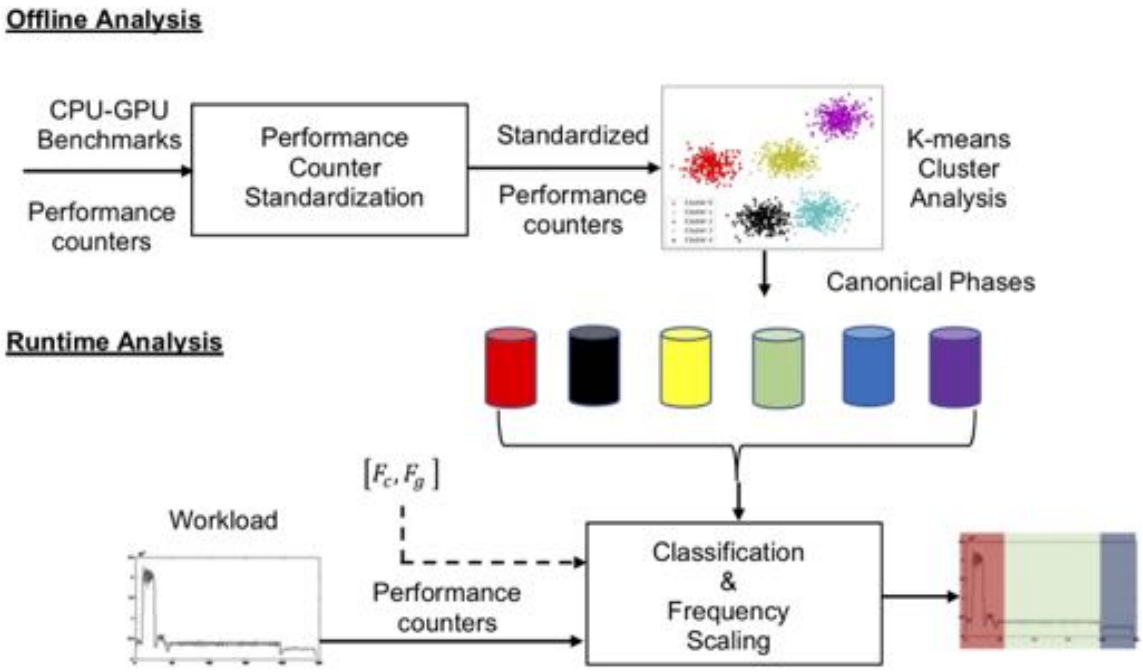

Figure 8.2: Workload-aware governor.

In offline analysis we aim to identify a set of CPs that can be used to detect the different workload phases. The different steps of the analysis are presented in Algorithm 2. As shown in Figure 8.2, we start by running a benchmark suite at different frequency levels, while collecting the following PC on the CPU side: instructions executed, branch-misses, cache misses and cache references, and on the GPU side we collect: GPU bus frequency and normalized GPU utilization.

Since the values of the PCs are a function of the DVFS setting, we need to standardize their values. We build a mean and a standard deviation models of the PCs using regression analysis, as shown in line 3, 4 and 5 of Algorithm 2. In line 6 the standardization is performed by subtracting the mean and dividing by the standard deviation. The GPU performance counters are normalized in regard of their maximum possible value. Finally, we conduct a k-means cluster analysis [74]. We identify six (i.e., $k = 6$) clusters (i.e., CPs), and compute their corresponding centroids as shown in line 7 of Algorithm 2. The number

Algorithm 2: Offline analysis of the WLA.

Output: C_{CP} CP centroids

Run benchmark suite at different frequency levels
PC $\leftarrow$ collect performance counters
$\overrightarrow{\mathrm{m}}$, $\overrightarrow{\mathrm{s}}$ $\leftarrow$ means and standard deviations of PCs
Mean PC model: $\overrightarrow{\mathrm{w_m}} = (\mathbf{F_c}^\intercal \mathbf{F_c})^{-1} \mathbf{F_c}^\intercal \overrightarrow{\mathrm{m}}$
Std PC model : $\overrightarrow{\mathrm{w_s}} = (\mathbf{F_c}^\intercal \mathbf{F_c})^{-1} \mathbf{F_c}^\intercal \overrightarrow{\mathrm{s}}$
$PC_s \leftarrow$ Standardize(PC,$\overrightarrow{\mathrm{w_m}}$, $\overrightarrow{\mathrm{w_s}}$)
$C_{CP} \leftarrow$ Apply *k*-means on PC_s and return centroids

of clusters was chosen such that each resulting canonical phase represents an actual workload phase. This is achieved by increasing the number of clusters, while keeping track of the time ratio of each CP when running different workloads. This time ratio represents the time spent on each CP, and it is referred to as the canonical phase composition (CPC).

For illustration, Table 8.1 gives the *canonical phase composition* (CPC) of various workloads. The table shows that the memory copy and memory latency workloads are mainly dominated by CP1, which means that CP1 corresponds to memory operation phases. The table also shows that all the remaining workloads are mainly dominated by the combination of CP1 and one of the remaining CP. Finally, we notice that Sling Shot, which is a GPU workload, is mainly dominated by CP4, which means that CP4 corresponds to a GPU workload.

8.2.2 Energy prediction model

In order to maximize the performance while meeting a target battery lifetime, for a set of CPU-GPU frequencies we need to be able to predict the energy consumption given the user usage history. For this goal, we represent the usage history through a user-specific CPC vector, which gives the percentage of time the user spends in each CP throughout the day. Each CPC vector represents the fractions of time spent by the user in each phase

Table 8.1: The canonical phase composition of different workloads.

	CP1 (%)	CP2 (%)	CP3 (%)	CP4 (%)	CP5 (%)	CP6 (%)
Ray Tracing	33	1	1	0	63	2
LLVM	31	4	56	0	4	5
Gaussian Blur	46	47	2	0	3	2
Speech Recognition	31	0	0	0	1	68
Memory Copy	96	0	1	0	3	0
Memory Latency	95	1	1	0	2	1
Sling Shot (GPU)	13	0	0	87	0	0

aggregated over workloads used throughout the day. Then, we use offline regression analysis to build an energy prediction model based on the CPs discussed earlier. The model is defined as follows:

$$EP(F,CPC) = T_r \sum_{i=1}^{i=k} (w_{ci} \cdot CPC_i \cdot Fc_i + w_{gi} \cdot CPC_i \cdot Fg_i) + c, \quad (8.1)$$

where F represents a vector of CPU and GPU frequency pairs (one pair per CP), Fc_i and Fg_i correspond to the CPU and GPU frequencies of the i^{th} CP, CPC_i is the i^{th} element of the CPC vector, T_r denotes the total runtime, k represents the number of CPs, and w_{ci}, w_{gi} and c are the weights to be determined by the regression analysis.

8.2.3 Frequency optimization

The frequency optimization aims to find the best CPU-GPU frequencies per CP. The optimization consists of two steps that rely on the energy prediction model to find the optimal frequency settings given the desired battery lifetime.

The CPU and the GPU combined have usually more than 30 DVFS levels, which makes the task of choosing the best settings per CP intractable. In order to solve this

Algorithm 3: Workload- and User-aware Battery Lifetime Management.

Input: C_{CP} (CP centroids), B_s (Battery status), B_c (Battery capacity), PET table, CPC (CP composition), EP (Energy prediction model)

1 E_r = $B_s.B_c$

2 Find F_{CP} (i.e, column in PET) such that:

$$\min |E_r - EP(F_{CP}, CPC)|$$

3 Increment or decrement F_{CP} such that:

$$EP(F_{CP}, CPC) \leq E_r$$
$$\nexists F' \; EP(F_{CP}, CPC) < EP(F', CPC) \leq E_r$$

4 **While** workload is running

5 $PC_s \leftarrow$ collect and standardize PCs

6 Find phase i that has the closest C_{CPi} to PC_s

7 $(Fc, Fg) \leftarrow F_{CPi}$

8 **end while**

efficiently, we construct offline a phase-aware performance-energy trade-off table (PET). The PET shown in Table 8.2 contains the CPU-GPU frequencies per CP that correspond to 5% decrements of the performance. It was obtained offline by running various benchmarks at different frequencies while keeping track of the performance.

The PET table is used within the runtime frequency optimization algorithm given in Algorithm 3 to identify the optimal DVFS settings. Algorithm 3 consists of three main steps.

First step (lines 1-2) This step aims to make a first estimation about the optimal DVFS settings per CP. It consists of going through the columns of the PET Table to find the column that has the closest energy to the remaining energy.

Second step (line 3) This step consists of doing frequency increments or decrements, to further minimize the difference between the expected energy consumption and the re-

Table 8.2: DVFS settings (MHz) of the phase-aware performance-energy trade-off table (PET).

	Performance 100%	Performance 95%	Performance 90%
CP1 (CPU, GPU)	(2112, 256)	(1958, 256)	(1804, 256)
CP2 (CPU, GPU)	(2342, 256)	(2208, 256)	(2112, 256)
CP3 (CPU, GPU)	(2361, 256)	(2265, 256)	(1958, 256)
CP4 (CPU, GPU)	(2342, 710)	(2208, 670)	(2035, 596)
CP5 (CPU, GPU)	(2342, 256)	(2112, 256)	(1881, 256)
CP6 (CPU, GPU)	(2342, 256)	(1958, 256)	(1804, 256)

maining battery energy. This is achieved by finding the setting F_{CP} that offers the smaller closest energy consumption to E_r. The procedure consists of choosing a CP and incrementing or decrementing its corresponding CPU and GPU frequency, which was obtained in the first step of the frequency optimization procedure. If from the first step we obtained $E_r > EP(F_{CP}, CPC)$, we increment the frequency, otherwise we decrement. After adjusting the frequency, we predict the new energy and check whether the equation in line 3 is satisfied, otherwise we repeat the same procedure by choosing a different CP. Throughout this procedure, the CPs are chosen based on their corresponding value in the CPC vector, such that the CP with smaller values are chosen first. The intuition is that CPs with smaller values are less executed by the user, thus, they allow small changes in the energy and performance.

Third step (lines 4-7) The optimal F_{CP} vector identified in the previous steps is given as input to the Workload-aware governor to assign the DVFS settings based on the current CP. As shown in line 5 to 7 of Algorithm 3, the PCs are continuously collected, and the classification is performed by measuring the Euclidean distance between the normalized PCs of the running workload, and the precomputed centroids of the chosen CPs from the offline analysis. The F_{CP} vector identified in the previous steps of Algorithm 3 contains frequency targets per CP, and is used after each classification to set the CPU and GPU

frequencies to level of the corresponding CP.

8.3 Experiments and results

8.3.1 Experimental setup

All the experiments are performed on an Android-based (Oreo version 8.0.0) Google Pixel 2 XL phone. The CPU and GPU performance are measured based on the multi-core score of Geekbench 4.3.1, and 3DMark scores. The Geekbench score is based on the run-time of each workload, the lower the run-time the higher the score is. The 3DMark scores are based on the frames per second (FPS), the higher the FPS the higher the score. Geekbench and 3DMark were chosen because they are by far the most widely used benchmarks in the mobile market by both companies and end-users. We used the Monsoon HV power monitor AAA10F to measure the power. The proposed work is implemented in C and runs on the actual smartphone. The management technique takes a DVFS decision each 20 ms, it has a minor memory footprint, and its computational overhead is less than 1%. Additionally, all the presented results already incorporate all computation overhead.

8.3.2 Results

Energy prediction model: In order to validate the prediction model, we collect the training data by running various workloads at different frequency levels. After performing the regression analysis, the accuracy of the energy prediction model was measured as compared to the actual energy values through several runs of workloads. The average percentage error across all runs is 7.3%.

We compare against state of the art battery lifetime management techniques, namely Powersave [42], BUSQ1 and BUSQ3 [51].

Powersave [42]: The used mobile platform allows a maximum CPU frequency of 2.45 GHz, and 710 MHz on the GPU side. When 20% of the capacity is reached, the maximum allowed CPU and GPU frequencies are decreased by 50% to 1.26 GHz and 342 MHz, respectively.

BUSQ1Ad [51]: In the case of a usage pattern that includes several stand-by phases, BUSQ1 ends up over-provisioning the available battery capacity, which results in a poor performance and a higher battery lifetime than required. For the purpose of a fair comparison, we evaluate our approach against a modified version of BUSQ1, referred to as BUSQ1Ad. The only difference as compared to BUSQ1, is that the linear discharge profile is steeper. Hence, the technique gets better performance, while closely meeting the user target lifetime.

BUSQ3 [51]: Rather than using a moving average to define the discharge profile for BUSQ3, we provide the exact discharge profile as input.

In order to perform an extensive evaluation, using two different benchmarks, we define two different user usage patterns[1]:

User Usage Pattern 1:

- $[0s, 500s]$ Geekbench multi-core (CPU)
- $[500s, 700s]$ Stand-by period

[1]Based on the energy consumption during these two patterns, we calculate the expected battery lifetime given the battery capacity of the used platform.

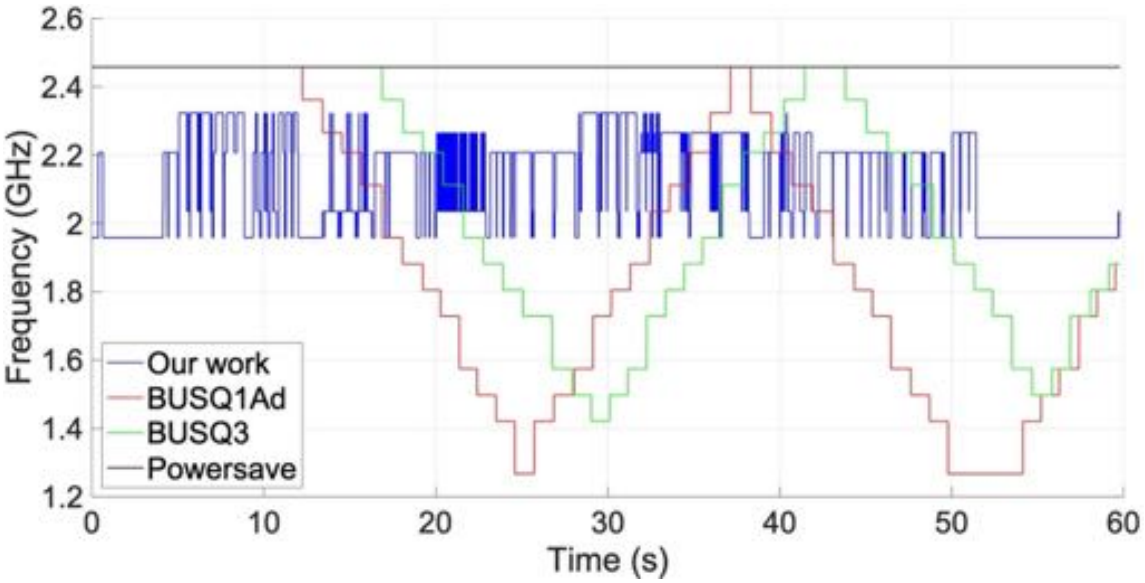

Figure 8.3: CPU frequency traces while running Geekbench.

- $[700s, 1200s]$ Geekbench multi-core (CPU)
- $[1200s, 2400s]$ Stand-by period

Figure 8.3 shows a portion of the CPU frequency traces of the implemented techniques. The frequency of BUSQ1Ad and BUSQ3 [51] starts at 2.457 GHz, afterwards their frequency is decremented until it reaches 1.267 GHz and 1.42 GHz, respectively. The frequency trace of Powersave [42] also starts at 2.457 GHz, but then as the battery capacity reaches 20% later in the experiment, the frequency is set to 1.267 GHz.

In the other hand, the energy prediction model allowed the proposed work to predict the optimal set of frequencies that can be maintained throughout the whole experiment. Figure 8.3 shows that the CPU frequency of the proposed work varies between 2.26 GHz and 1.95 GHz, by switching between 5 frequency levels, which means that the shown workload contains 5 CP phases. As shown in Table 8.3, this leads to a better performance, reflected through a higher score, less QoS variation and a longer battery lifetime. As compared to BUSQ3 [51], the proposed work is showing an improvement of 5.4% and 6.1% on

Table 8.3: CPU evaluation of the proposed technique using Geekbench4 (Higher scores mean better performance).

	Crypto	Integer	Float	Memory	Total	QoS Variation	Battery Life (h)
Our work	5148 -3.7%	7704 5.4%	5344 6.1%	2476 -1.5%	5823 6.1%	1.9%	8.6 2.5%
Powersave[42]	4944 -7.5%	7501 2.6%	5064 6.4%	2426 -3.5%	5627 2.5%	62%	8.04 -4.2%
BUSQ1Ad[51]	5022 -6.1%	7200 -1.5%	4655 -2.2%	2218 -11.8%	5331 -2.9%	23%	8.49 1.2%
BUSQ3[51]	5348 0%	7310 0%	4761 0%	2514 0%	5488 0%	14%	8.39 0%

the Integer and Floating-Point sections, respectively. The total performance improvement is 6.1% with 2.5% improvement in the battery lifetime, as compared to BUSQ3. The QoS variation, which represents the percentage difference between the best and the worst score throughout the experiment, shows that the proposed work has only 1.88% performance variation, which is 7x less performance variation as compared BUSQ3.

User Usage Pattern 2:

- $[0s, 950s]$ 3DMark Sling Shot (CPU-GPU)
- $[950s, 1550s]$ Stand-by period
- $[1550s, 2500s]$ 3DMark Sling Shot (CPU-GPU)
- $[2500s, 5300s]$ Stand-by period

Figure 8.4 shows a portion of the GPU frequency traces of the implemented techniques. The frequency of BUSQ1Ad and BUSQ3 [51] start at 710 MHz, then it is decremented until it reaches 257 MHz and 414 MHz, respectively. For Powersave the frequency starts at 710 MHz, but then as the battery capacity reaches 20% later in the experiment, it is set to 342 MHz.

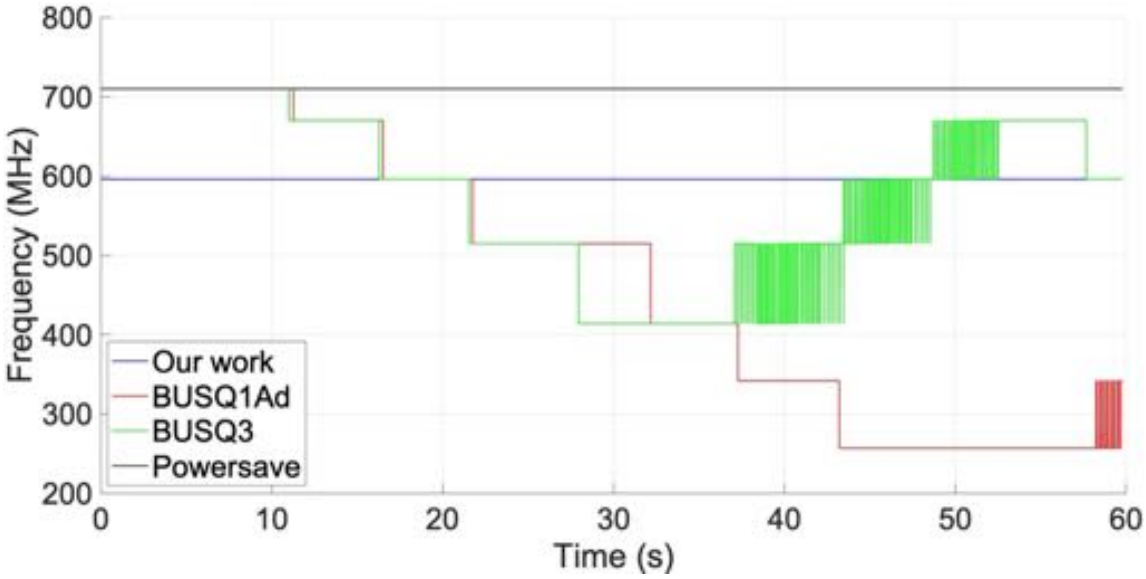

Figure 8.4: GPU frequency traces while running 3DMark.

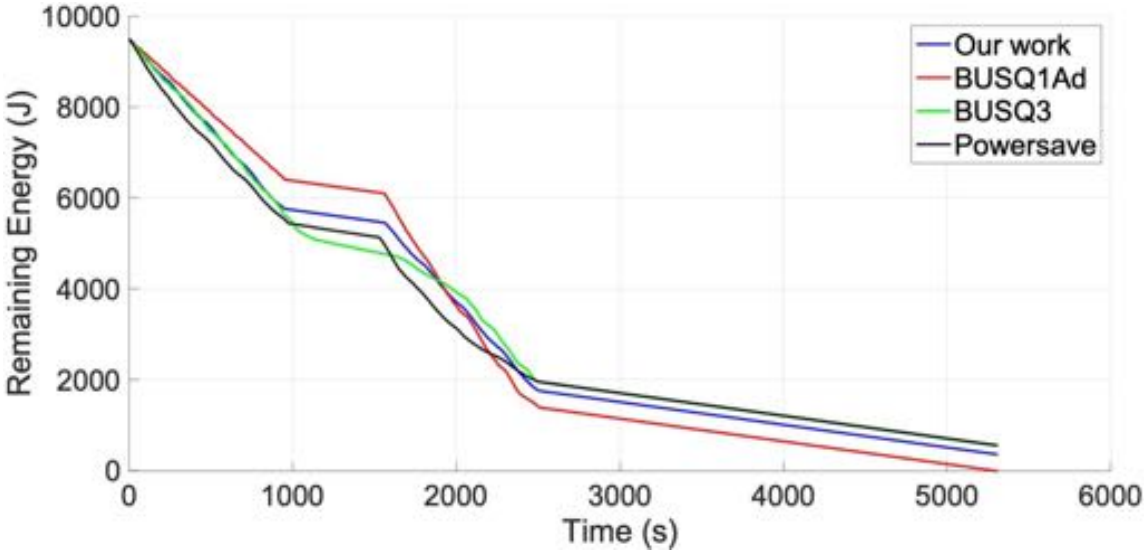

Figure 8.5: Discharge profile.

Table 8.4: CPU-GPU evaluation of the proposed technique using 3DMark (Higher scores mean better performance).

	Graphics score	Physics score	Total score	QoS Variation	Battery Life (h)
Our work	4780 (9.4%)	2718 (15.8%)	4089 (11.9%)	3.6%	8.53 (0%)
Powersave[42]	4415 (1.5%)	2484 (5.8%)	3760 (2.9%)	50%	8.85 (3.4%)
BUSQ1Ad[51]	4578 (5.3%)	2624 11.8%)	3882 (6.3%)	38%	8.2 (-4.2%)
BUSQ3[51]	4348 (0%)	2384 (0%)	3653 (0%)	55.4%	8.56 (0%)

In the other hand, the GPU frequency of the proposed work in Figure 8.4 is stable at 596 MHz. The 60 seconds portion showed in Figure 8.4 is mainly a GPU workload phase, corresponding to CP4, whose optimal frequency was predicted to be 596 MHz. Figure 8.5 shows the battery discharge profile of our work as compared to the other techniques while running the usage pattern 2. The figure shows that the discharge profile of Powersave [42] drowns pretty fast in the first 1700 seconds, then the energy consumption slows down significantly. In contrast to Powersave, in the first 1550 seconds, BUSQ1Ad is over-conservative in energy consumption, then the remaining energy drowns rapidly. On the other hand, our technique maintains a stable energy consumption profile. As shown in Figure 8.5, the energy profile of the proposed work is at all times confined between the under-conservative and the over-conservative discharge profiles. Hence, providing an optimal and seamless performance, while meeting the target lifetime. This is reflected through Table 8.4 that shows a better performance, less QoS variation and a longer battery lifetime. As compared to BUSQ3 [51], our work is showing an improvement of 9.4% and 15.8% on the graphics and physics sections of 3DMark, respectively, with a total performance improvement of 11.9%.

Additionally, our work has only 3.6% performance variation, which is 15$\times$ less performance variation as compared to BUSQ3. This is demonstrated through Figure 8.6, that

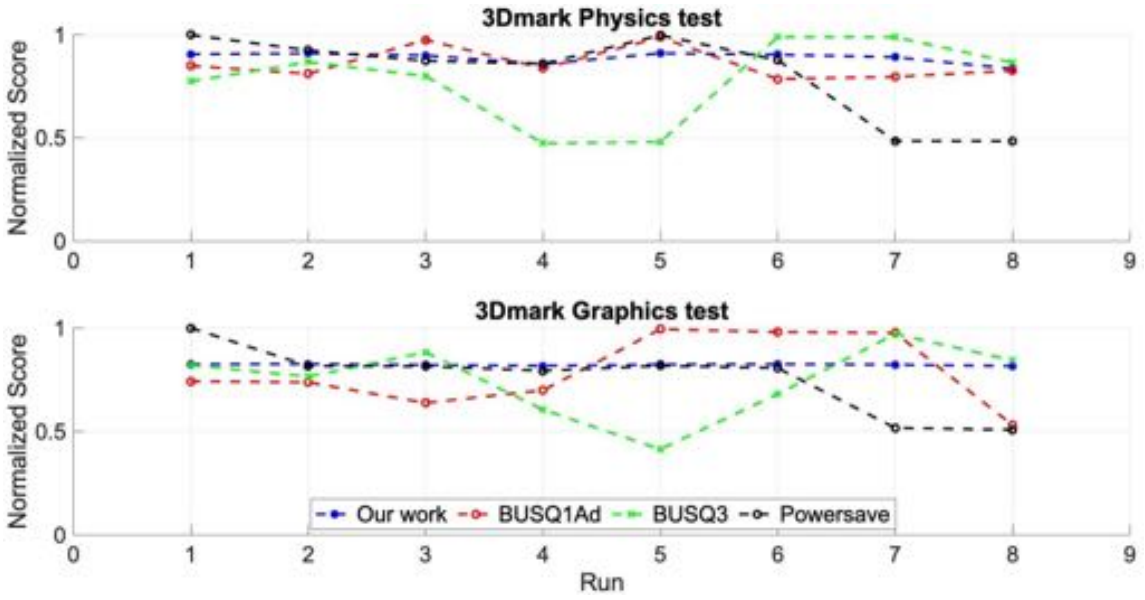

Figure 8.6: QoS variation using 3DMark.

shows the normalized physics and graphics scores of 8 runs of 3DMark, divided by a 10 minutes break after the fourth run. The figure shows that the proposed work offers a seamless and stable performance on CPU and GPU side, as compared to the other techniques, whose normalized score varies between 0.4 and 1.

8.4 Conclusion

This chapter investigated a workload- and user-aware battery lifetime management technique for Mobile SoCs. The proposed technique manages both the CPU and the GPU to maximize performance while meeting a target battery lifetime. During an offline analysis we design a Workload-aware governor by identifying a set of canonical workload phases using cluster analysis. We build an energy prediction model based on the canonical phase composition of the user usage history. During runtime, a frequency optimization procedure uses the energy prediction model to find the optimal set of frequencies per canonical

phase. Finally, the Workload-aware governor identifies the phase of the current workload and uses the frequencies per canonical phase to scale the frequency. The proposed technique achieves up to 11.9% better performance and it decreases the performance variation by 10× while meeting the user desired battery lifetime.

Chapter 9

Summary and Possible Extensions

This thesis aims at improving the performance and the battery lifetime of mobile devices through thermal and power sensing and management. On the sensing side, we proposed the Alternating-BPI technique that accurately estimates the power consumption of individual hardware units without the use of any design based models. Additionally, we proposed a Deconvolutional Neural Network (DCNN) based power map estimation. The Alternating-BPI technique was then used to perform a power and hardware characterization of Augmented Reality Apps. For runtime management, we proposed a coordinated self-tuning thermal management controller based on online deep learning. In addition, we proposed a cascaded controller for mobile devices to manage the different sources of current and thermal emergencies, while maximizing performance. Finally, we designed a novel workload- and user-aware battery lifetime management technique that maximizes the performance under the user's desired battery lifetime. Section 9.1 summarizes our contributions. We discuss potential future extensions in Section 9.2.

9.1 Summary of the Dissertation

In Chapter 3, we investigated the lack of fine-grain power sensing in modern SoCs, and proposed an Alternating Blind Identification of Power sources (Alternating-BPI), a technique that accurately estimates the power consumption of individual SoC units by relying on the measurements from the embedded thermal sensors and the total power consumption. The accuracy and applicability of the proposed technique was verified using simulation and experimental data. We showed that Alternating-BPI is able to estimate the power at the SoC hardware unit level with up to 98.1% accuracy. Furthermore, we demonstrate the applicability of the proposed technique on a commercial SoC.

In Chapter 4, We proposed to solve the power map estimation problem as an image generation problem using Deconvolutional Neural Networks (DCNN). The proposed DCNN takes as input the thermal measurements from the embedded thermal sensors to estimate the full SoC power map. The proposed technique allows to estimate the power map at a finer spatial granularity than the thermal spatial granularity of the existing thermal sensors. More specifically, it allows to estimate the power even at locations where thermal measurements are not physically available. The proposed technique is demonstrated using a commercial SoC while running several benchmarks. The predicted power maps show a 97% similarity (2D correlation) with the power maps estimated using the Alternating-BPI.

In Chapter 5, we performed a power and hardware characterization of Augmented Reality Applications. After performing an analysis using existing AR Apps, we designed and developed ARBench, an augmented reality benchmark for mobile devices. Then, ARBench was used to evaluate the AR performance of existing commercial mobile devices, and to perform a phase analysis using performance counters to characterize existing AR Apps. Finally, the benchmark was used to study the performance and power trade-offs

of different CPU multi-core configurations, and we provide insights that could be used to save power while meeting the AR performance requirements.

In Chapter 6, we investigated few challenges related to thermal management on mobile devices, including the workload and the ambient temperature dependency of the optimal PID parameters, in addition to the challenge of controlling multiple sources of thermal emergency. This investigation led us to propose a coordinated self-tuning thermal management controller, that relies on online deep learning to continuously adapt to characteristics and operating conditions. The proposed controller takes into account both skin temperature and junction temperature constraints in a coordinated manner. We implemented the controller on a commercial smartphone, and we evaluate it comprehensively against over techniques from the literature, under different ambient temperatures and workload characteristics. Our results demonstrate that our coordinated self-tuning thermal controller leads to 6% better performance, and spends up to 27$\times$ less time in thermal violation.

In Chapter 7, we investigated the fact that existing techniques dissociate the current throttling, the junction temperature throttling and the skin temperature throttling by controlling each measure separately, which leads to sub-optimal control. Then we design a cascaded controller (CasCon) that manages the skin temperature, the junction temperature and the electrical current in a coordinated manner. The proposed controller dynamically changes the thermal and electrical current caps in runtime, allowing it to save power and improve performance. We also introduce a frequency locking function that significantly reduces the number of DVFS transitions, hence bringing extra power savings. We implement our CasCon controller on a real smartphone and we evaluate it comprehensively at different ambient temperatures. Our results show that the proposed controller successfully prevents current and thermal violations even at high ambient temperatures, while bringing up to 6.5% performance improvements and 18% power savings compared to previous work.

In Chapter 8, we argue that the optimal balance between performance and battery lifetime depends on the user desired target battery lifetime. Hence, we propose a workload- and user-aware battery lifetime management technique that maximizes the performance under the constraint of the user target battery lifetime. The workload-awareness is achieved through performance counters, while the user awareness is incorporated by representing the user-usage history through a set of canonical phases (CP). In order to achieve the best results, we had to design a novel model that predicts the energy consumption based on the user usage history. Then, the proposed battery lifetime management is achieved by scaling both the CPU and the GPU DVFS levels, unlike previous techniques that do not consider the GPU. We implemented our technique on a commercial smartphone and compared its performance against state-of-the-art battery management techniques. The proposed technique achieved 15.8% and 9.4% performance improvement on the CPU and GPU, respectively, while meeting the lifetime target and decreasing the performance variation by 10x.

9.2 Possible Research Extensions

There are several opportunities to improve thermal and power sensing and management for Mobile SoCs. There are four natural extensions to our presented work in this thesis. First, we showed that using the internal thermal sensors and the total power, we can predict the full SoC power map by relying on the Alternating-BPI and a deconvolutional neural network. This work could be extended to SoCs that lack thermal sensors, by building a model that predicts the full power map of the SoC based on performance counters. Second, the power characterization of Augmented Reality Apps has shown that they are power greedy, with more than half the power being consumed by the mapping and tracking algorithms. Thus, an interesting track would be to explore energy-efficient computation off-loading

of Augmented Reality processes to extend the battery lifetime of AR/VR headsets, under latency and accuracy constraints. Third, the proposed work in this thesis, allowing the prediction of the full power map of the SoC could be extended to predict, locate and prevent the thermal hotspots through runtime management, even for locations on the SoC that do not have physical thermal sensors. Finally, the proposed workload- and user-aware battery lifetime management aims to maximize the performance given a target battery lifetime, assuming that the user would be able to correctly estimate and provide as input the target battery lifetime. Accurately estimating the target battery lifetime would help in greatly improving the user experience, because this would enable a more optimal trade-off between performance and battery lifetime. Thus, an extension to this work would be to remove the user from the loop, by designing a machine learning algorithm able to estimate the target battery lifetime, based on the user historical data like the date, time, and location the previous battery recharges.

Bibliography

[1] 3dmark gpu benchmark. https://www.3dmark.com/.

[2] Ai benchmark. https://ai-benchmark.com/.

[3] Ar 3d animals - apps on google play. https://play.google.com/store/apps/details?id=com.grappsgames.

[4] Ar dragon - apps on google play. https://play.google.com/store/apps/details?id=com.laugh.ar.dragon.

[5] Augmently – augmented reality for furniture - apps on google play. https://play.google.com/store/apps/details?id=com.Audaxlabs.AudaxARView.

[6] Civilisations ar. https://play.google.com/store/apps/details?id=uk.co.bbc.civilisations&hl=en_US&gl=US.

[7] cross browser testing platform. https://www.browserstack.com/.

[8] Geekbench 4. https://www.geekbench.com/geekbench4/.

[9] Hello ar core; google developers. https://developers.google.com/ar/develop/unity/tutorials/hello-ar-sample.

[10] High voltage power monitor: Monsoon solutions: Bellevue. https://www.msoon.com/high-voltage-power-monitor.

[11] Knightfall ar - apps on google play. https://play.google.com/store/apps/details?id=com.aetn.games.knightfall.ar.

[12] Mission to mars ar - apps on google play. https://play.google.com/store/apps/details?id=com.sndigital.marsar.

[13] Monster park ar - jurassic dinosaurs in real world - apps on google play. https://play.google.com/store/apps/details?id=com.vitotechnology.DinoAR.

[14] Qualcomm snapdragon 865 specs. https://www.androidauthority.com/qualcomm-snapdragon-865-specs-1058483/.

[15] Samsung ar - apps on google play. https://play.google.com/store/apps/details?id=com.vrai.samsungsurfaceAR.

[16] Scale lab tools. https://scale.engin.brown.edu/software/.

[17] Sketchar create art and get nft instantly - apps on google play. https://play.google.com/store/apps/details?id=ktech.sketchar.

[18] Snapdragon 865 mobile hardware development kit. https://developer.qualcomm.com/hardware/snapdragon-865-hdk.

[19] Snapdragon 865 mobile hardware development kit. https://developer.qualcomm.com/hardware/snapdragon-865-hdk.

[20] Snapdragon profiler. https://developer.qualcomm.com/software/snapdragon-profiler.

[21] Vrmark - the virtual reality benchmark. https://benchmarks.ul.com/vrmark.

[22] Youcam makeup - selfie editor - apps on google play. https://play.google.com/store/apps/details?id=com.cyberlink.youcammakeup.

[23] Nick Baker, Marco Liserre, Laurent Dupont, and Yvan Avenas. Improved reliability of power modules: A review of online junction temperature measurement methods. *IEEE Industrial Electronics Magazine*, 8(3):17–27, 2014.

[24] Nilanjan Banerjee, Ahmad Rahmati, Mark D Corner, Sami Rollins, and Lin Zhong. Users and batteries: interactions and adaptive energy management in mobile systems. In *International conference on ubiquitous computing*, pages 217–234. Springer, 2007.

[25] Andrea Bartolini, Matteo Cacciari, Andrea Tilli, and Luca Benini. A distributed and self-calibrating model-predictive controller for energy and thermal management of high-performance multicores. In *2011 Design, Automation & Test in Europe*, pages 1–6. IEEE, 2011.

[26] Andrea Bartolini, Matteo Cacciari, Andrea Tilli, and Luca Benini. Thermal and energy management of high-performance multicores: Distributed and self-calibrating model-predictive controller. *IEEE Transactions on Parallel and Distributed Systems*, 24(1):170–183, 2013.

[27] Francesco Beneventi, Andrea Bartolini, Andrea Tilli, and Luca Benini. An effective gray-box identification procedure for multicore thermal modeling. *IEEE Transactions on Computers*, 63(5):1097–1110, 2012.

[28] David Brooks, Vivek Tiwari, and Margaret Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. *ACM SIGARCH Computer Architecture News*, 28(2):83–94, 2000.

[29] Huifeng Chen, Bing Ji, Volker Pickert, and Wenping Cao. Real-time temperature estimation for power mosfets considering thermal aging effects. *IEEE Transactions on Device and Materials Reliability*, 14(1):220–228, 2013.

[30] Huixiang Chen, Yuting Dai, Hao Meng, Yilun Chen, and Tao Li. Understanding the characteristics of mobile augmented reality applications. In *2018 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 128–138. IEEE, 2018.

[31] Sofiane Chetoui and Sherief Reda. Coordinated self-tuning thermal management controller for mobile devices. *IEEE Design & Test*, 2020.

[32] Ui-Min Choi, Frede Blaabjerg, and Søren Jørgensen. Study on effect of junction temperature swing duration on lifetime of transfer molded power igbt modules. *IEEE Transactions on Power Electronics*, 32(8):6434–6443, 2017.

[33] Yonghun Choi, Seonghoon Park, and Hojung Cha. Graphics-aware power governing for mobile devices. In *Proceedings of the 17th Annual International Conference on Mobile Systems, Applications, and Services*, pages 469–481, 2019.

[34] Ryan Cochran, Can Hankendi, Ayse K Coskun, and Sherief Reda. Pack & cap: adaptive dvfs and thread packing under power caps. In *2011 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 175–185. IEEE, 2011.

[35] Ryan Cochran and Sherief Reda. Consistent runtime thermal prediction and control through workload phase detection. In *Design Automation Conference*, pages 62–67. IEEE, 2010.

[36] Ryan Cochran and Sherief Reda. Thermal prediction and adaptive control through workload phase detection. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 18(1):7, 2013.

[37] Howard David, Eugene Gorbatov, Ulf R Hanebutte, Rahul Khanna, and Christian Le. Rapl: Memory power estimation and capping. In *2010 ACM/IEEE International Symposium on Low-Power Electronics and Design (ISLPED)*, pages 189–194. IEEE, 2010.

[38] Da Deng. Li-ion batteries: basics, progress, and challenges. *Energy Science & Engineering*, 3(5):385–418, 2015.

[39] Kapil Dev, Abdullah Nazma Nowroz, and Sherief Reda. Power mapping and modeling of multi-core processors. In *International Symposium on Low Power Electronics and Design (ISLPED)*, pages 39–44. IEEE, 2013.

[40] Brad K Donohoo, Chris Ohlsen, and Sudeep Pasricha. Aura: An application and user interaction aware middleware framework for energy optimization in mobile devices. In *29th International Conference on Computer Design (ICCD)*, pages 168–174. IEEE, 2011.

[41] Begum Egilmez, Gokhan Memik, Seda Ogrenci-Memik, and Oguz Ergin. User-specific skin temperature-aware dvfs for smartphones. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2015*, pages 1217–1220. IEEE, 2015.

[42] Utkarsh Goel, Stephen Ludin, and Moritz Steiner. Web performance with android's battery-saver mode. *arXiv preprint arXiv:2003.06477*, 2020.

[43] Young-Ho Gong, Jae Jeong Yoo, and Sung Woo Chung. Thermal modeling and validation of a real-world mobile ap. *IEEE Design & Test*, 35(1):55–62, 2017.

[44] Yin Hang and Hussameddine Kabban. Thermal management in mobile devices: challenges and solutions. In *2015 31st Thermal Measurement, Modeling & Management Symposium (SEMI-THERM)*, pages 46–49. IEEE, 2015.

[45] Yao He, XingTao Liu, ChenBin Zhang, and ZongHai Chen. A new model for state-of-charge (soc) estimation for high-power li-ion batteries. *Applied Energy*, 101:808–814, 2013.

[46] Chen-Ying Hsieh, Jurn-Gyu Park, Nikil Dutt, and Sung-Soo Lim. Memory-aware cooperative cpu-gpu dvfs governor for mobile games. In *13th Symposium on Embedded Systems For Real-time Multimedia (ESTIMedia)*, pages 1–8. IEEE, 2015.

[47] Wei Huang, Shougata Ghosh, Sivakumar Velusamy, Karthik Sankaranarayanan, Kevin Skadron, and Mircea R Stan. Hotspot: A compact thermal modeling methodology for early-stage vlsi design. *IEEE Transactions on very large scale integration (VLSI) systems*, 14(5):501–513, 2006.

[48] Jae Min Kim, Young Geun Kim, and Sung Woo Chung. Stabilizing cpu frequency and voltage for temperature-aware dvfs in mobile devices. *IEEE Transactions on Computers*, 64(1):286–292, 2013.

[49] Yeseong Kim, Pietro Mercati, Ankit More, Emily Shriver, and Tajana Rosing. P 4: Phase-based power/performance prediction of heterogeneous systems via neural networks. In *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 683–690. IEEE, 2017.

[50] Daniel D Lee and H Sebastian Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401(6755):788–791, 1999.

[51] Wooseok Lee, Reena Panda, Dam Sunwoo, Jose Joao, Andreas Gerstlauer, and Lizy K John. Buqs: battery-and user-aware qos scaling for interactive mobile devices. In *23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 64–69. IEEE, 2018.

[52] Duo Li, Sheldon X-D Tan, Eduardo H Pacheco, and Murli Tirumala. Parameterized architecture-level dynamic thermal models for multicore microprocessors. *ACM*

Transactions on Design Automation of Electronic Systems (TODAES), 15(2):1–22, 2010.

[53] Xueliang Li, Guihai Yan, Yinhe Han, and Xiaowei Li. Smartcap: user experience-oriented power adaptation for smartphone's application processor. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 57–60. IEEE, 2013.

[54] Min Yeol Lim, Allan Porterfield, and Robert Fowler. Softpower: fine-grain power estimations using performance counters. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, pages 308–311, 2010.

[55] Rajiv Mongia, A Bhattacharya, and Himanshu Pokharna. Skin cooling and other challenges in future mobile form factor computing devices. *Microelectronics Journal*, 39(7):992–1000, 2008.

[56] A Munier, JR Burgan, J Gutierrez, E Fijalkow, and MR Feix. Group transformations and the nonlinear heat diffusion equation. *SIAM Journal on Applied Mathematics*, 40(2):191–207, 1981.

[57] Farid N Najm. Power estimation techniques for integrated circuits. In *Proceedings of IEEE International Conference on Computer Aided Design (ICCAD)*, pages 492–499. IEEE, 1995.

[58] Cameron Nelson and Jesse Galloway. Package thermal challenges due to changing mobile system form factors. In *2018 34th Thermal Measurement, Modeling & Management Symposium (SEMI-THERM)*, pages 98–106. IEEE, 2018.

[59] Edson Luiz Padoin, Laércio Lima Pilla, Márcio Castro, Francieli Z Boito, Philippe Olivier Alexandre Navaux, and Jean-François Méhaut. Performance/energy trade-off in scientific computing: the case of arm big. little and intel sandy bridge. *IET Computers & Digital Techniques*, 9(1):27–35, 2015.

[60] Jihoon Park, Seokjun Lee, and Hojung Cha. Accurate prediction of smartphones' skin temperature by considering exothermic components. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1500–1503. IEEE, 2018.

[61] Jurn-Gyu Park, Nikil Dutt, Hoyeonjiki Kim, and Sung-Soo Lim. Hicap: Hierarchical fsm-based dynamic integrated cpu-gpu frequency capping governor for energy-efficient mobile gaming. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 218–223, 2016.

[62] Nadja Peters, Sangyoung Park, Daniel Clifford, Sami Kyostila, R Mcllroy, Benedikt Meurer, Hannes Payer, and Samarjit Chakraborty. Phase-aware web browser power management on hmp platforms. In *Proceedings of the International Conference on Supercomputing*, pages 274–283, 2018.

[63] Emirhan Poyraz and Gokhan Memik. Using built-in sensors to predict and utilize user satisfaction for cpu settings on smartphones. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 3(1):1–25, 2019.

[64] Alok Prakash, Hussam Amrouch, Muhammad Shafique, Tulika Mitra, and Jörg Henkel. Improving mobile gaming performance through cooperative cpu-gpu thermal management. In *Proceedings of the 53rd Annual Design Automation Conference*, page 47. ACM, 2016.

[65] Robin Randhawa. Software techniques for arm big. little systems. *ARM, Apr*, 2013.

[66] Sherief Reda and Adel Belouchrani. Blind identification of power sources in processors. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, pages 1739–1744. IEEE, 2017.

[67] Sherief Reda, Kapil Dev, and Adel Belouchrani. Blind identification of thermal models and power sources from thermal measurements. *IEEE Sensors Journal*, 18(2):680–691, 2017.

[68] Sherief Reda, Abdullah N Nowroz, Ryan Cochran, and Stefan Angelevski. Post-silicon power mapping techniques for integrated circuits. *Integration*, 46(1):69–79, 2013.

[69] Sheriff Sadiqbatcha, Yue Zhao, Jinwei Zhang, Hussam Amrouch, Jörg Henkel, and Sheldon X-D Tan. Machine learning based online full-chip heatmap estimation. In *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 229–234. IEEE, 2020.

[70] Onur Sahin and Ayse K Coskun. On the impacts of greedy thermal management in mobile devices. *IEEE Embedded Systems Letters*, 7(2):55–58, 2015.

[71] Onur Sahin, Lothar Thiele, and Ayse K Coskun. Maestro: Autonomous qos management for mobile applications under thermal constraints. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2018.

[72] Mostafa Said, Sofiane Chetoui, Adel Belouchrani, and Sherief Reda. Understanding the sources of power consumption in mobile socs. In *2018 Ninth International Green and Sustainable Computing Conference (IGSC)*, pages 1–7. IEEE, 2018.

[73] Krishna Sekar. Power and thermal challenges in mobile devices. In *Proceedings of the 19th annual international conference on Mobile computing & networking*, pages 363–368, 2013.

[74] Shokri Z Selim and Mohamed A Ismail. K-means-type algorithms: A generalized convergence theorem and characterization of local optimality. *Transactions on pattern analysis and machine intelligence*, (1):81–87, 1984.

[75] Karan Singh, Major Bhadauria, and Sally A McKee. Real time power estimation and thread scheduling via performance counters. *ACM SIGARCH Computer Architecture News*, 37(2):46–55, 2009.

[76] Gaurav Singla, Gurinderjit Kaur, Ali K Unver, and Umit Y Ogras. Predictive dynamic thermal and power management for heterogeneous mobile platforms. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, pages 960–965. EDA Consortium, 2015.

[77] Kevin Skadron, Mircea Stan, Marco Barcella, Amar Dwarka, Wei Huang, Yingmin Li, Yong Ma, Amit Naidu, Dharmesh Parikh, Paolo Re, et al. Hotspot: Techniques for modeling thermal effects at the processor-architecture level. Citeseer.

[78] Yefu Wang, Kai Ma, and Xiaorui Wang. Temperature-constrained power control for chip multiprocessors with online model estimation. *ACM SIGARCH computer architecture news*, 37(3):314–324, 2009.

[79] Qing Xie, Jaemin Kim, Yanzhi Wang, Donghwa Shin, Naehyuck Chang, and Massoud Pedram. Dynamic thermal management in mobile devices considering the thermal coupling between battery and application processor. In *Computer-Aided Design (ICCAD), 2013 IEEE/ACM International Conference on*, pages 242–247. IEEE, 2013.

[80] Fei Xu, Shuai Yang, Zhi Zhou, and Jia Rao. ebrowser: Making human-mobile web interactions energy efficient with event rate learning. In *38th International Conference on Distributed Computing Systems (ICDCS)*, pages 523–533. IEEE, 2018.

[81] Kaige Yan, Xingyao Zhang, and Xin Fu. Characterizing, modeling, and improving the qoe of mobile devices with low battery level. In *2015 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 713–724. IEEE, 2015.

[82] Masaki Yoshio, Ralph J Brodd, and Akiya Kozawa. *Lithium-ion batteries*, volume 1. Springer, 2009.

[83] Ying-Ju Yu and Carole-Jean Wu. Understanding the thermal challenges of high-performance mobile devices with a detailed platform temperature model. In *2017 IEEE International Symposium on Workload Characterization (IISWC)*, pages 122–123. IEEE, 2017.

[84] John G Ziegler and Nathaniel B Nichols. Optimum settings for automatic controllers. *trans. ASME*, 64(11), 1942.

[85] Ghassan Zubi, Rodolfo Dufo-López, Monica Carvalho, and Guzay Pasaoglu. The lithium-ion battery: State of the art and future perspectives. *Renewable and Sustainable Energy Reviews*, 89:292–308, 2018.